ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA INFORMÁTICA
GRADO EN INGENIERÍA DEL SOFTWARE

SIMULACIÓN EN TIEMPO ACELERADO DE UNA RED DE AUTOBUSES METROPOLITANA USANDO UN CAS

(ACCELERATED-TIME SIMULATION OF A CITY BUS NETWORK USING A CAS)

Realizado por
JOSÉ MANUEL GAVILÁN MONCADA
Tutorizado por
Dr. D. GABRIEL AGUILERA VENEGAS Y
Dr. D. JOSÉ LUIS GALÁN GARCÍA
Departamento
MATEMÁTICA APLICADA

UNIVERSIDAD DE MÁLAGA
MÁLAGA, Diciembre 2014

Fecha defensa:
El Secretario del Tribunal

Agradecimientos

A Gabriel y José Luis por el trato increíble que he recibido durante la realización de este proyecto, confianza depositada y grandes consejos.

A mis amigos y a mi familia en general, pero por encima de todo quiero dar las gracias a las cuatro personas que les debo todo lo que tengo y soy.

A mi madre Mari Carmen y a mi tía Divina, porque aunque dicen que madre solo hay una, yo tengo dos.

A mi padre Juan, que aunque el destino no quiso que pasáramos mucho tiempo juntos fue más que suficiente para ver lo increíble que eras.

Y en último lugar como mención especial a mi tío Salvador, porque él ha hecho algo que la gente normal no puede hacer, él ha podido elegir quien es su otro hijo, y yo he tenido la suerte de ser su elegido. Muchísimas gracias por todo.

Resumen:

Se ha realizado una aplicación para simular en tiempo acelerado una red de autobuses metropolitana y de metro usando un CAS. Con esta aplicación se pretende optimizar estos medios de transporte de forma que el uso de estos sea la primera posibilidad a elegir y no la última alternativa. Esta aplicación permite la representación de cualquier mapa descrito por el usuario, para ello se han desarrollado una serie de algoritmos capaces de representarlos y registrar la información necesaria para el correcto movimiento de los autobuses y del metro. La generación de personas en las paradas de autobuses y metro, modificación del tiempo que se tarda en recorrer la distancia entre dos paradas en función del tráfico que exista y la generación de posibles averías tanto en los autobuses como en los metros se ha realizado usando funciones de distribución como la Exponencial, Poisson y Normal. Estas funciones varían dependiendo de los parámetros que se introduzcan mediante la interfaz de usuario. El CAS encargado de realizar la simulación es Maxima y tanto para la representación gráfica de la simulación como para mostrar los resultados se ha usado JAVA.

Palabras claves: Simulación Java Maxima CAS Jacomax Variables aleatorias Funciones de distribución Tiempo acelerado Autobuses Metro

Abstract:

An application was developed in order to simulate a metropolitan bus and subway net at an accelerated time by using a CAS. With this application we pretend to optimize these means of transport so that their use is the first possibility to choose and not the last alternative. This application allows the representation of any map described by the user. For this, a kind of algorithm able to represent them and to register the necessary information for the right movement of the buses and the subway was developed. The generation of people at the bus and subway stops, the modification of the time it takes to do the distance between two stops depending on the traffic at that moment, and the generation of potential breakdowns both of the buses and the subway were made by using the distribution functions, such as the exponential, Poisson and Normal. These functions vary depending on the parameters introduced by means of the user's interface. The CAS responsible for doing the simulation is Maxima, and JAVA was used for both the graphic representation of the simulation and to show the results.

Keywords: Simulation Java Maxima CAS Jacomax Accelerated-time Random variable Distribution fuctions Buses Subway

Índice general

1 Introducción

La finalidad de este capítulo es introducir al lector el problema que pretendemos abordar y como pretendemos hacerlo.

1.1 Motivación

El transporte público en general y en particular las líneas de autobuses urbanas y el metro permiten el desplazamiento de personas de un punto a otro en el área de una ciudad y es, por tanto, parte esencial de las ciudades.
Al mismo tiempo, las líneas de autobuses urbanas y el metro además de fomentar una forma plácida y agradable de viajar, evitan en gran medida las retenciones, atascos y los posibles errores originados por un desconocimiento de la zona.

Por esto y más motivos el uso del metro y autobuses públicos supone la alternativa más ecológica y solidaria para muchos de los desplazamientos que se hacen dentro de las ciudades. Tristemente las estadísticas dicen que hoy en día sólo usa estos transportes quien no tiene alternativa, esto se debe muchas veces a que no suele encontrarse una línea que se pueda adecuar a las necesidades de trabajo, estudios u ocio.

Es por todo esto por lo que se intenta hacer que el tráfico de autobuses y metro sea lo más óptimo posible para así lograr un mayor uso de estos. Pero resulta difícil probar todas las distintas combinaciones de líneas y/o paradas de forma empírica. Para facilitar el llevar a cabo esta tarea se ha realizado este TFG.

1.2 Objetivos del trabajo

Inicialmente el objetivo del presente trabajo era diseñar una aplicación que fuese capaz de simular una red de autobuses metropolitana.

Pero una vez comenzado el trabajo se pensó que se le podía dar una utilidad más, esta utilidad más era hacer que las simulaciones puedan combinar líneas de metro con líneas de autobuses. Esto es posible debido a que a efectos prácticos los datos que definen a una línea de metro y a una línea de autobuses son los mismos.

Añadiendo a la simulación la posibilidad del metro la aplicación está capacitada para poder simular el funcionamiento de los dos grandes medios de transporte público que existen hoy en día en las ciudades, los autobuses y el metro, y no solo las líneas de autobuses como se pretendía inicialmente.

En la simulación, el movimiento de los distintos vehículos se generará de forma automática usando un CAS (siglas de Computer Algebra System,

que significa sistema automático de cálculo simbólico) concretamente en MAXIMA.

Cabe destacar el paralelismo entre esta idea y las desarrolladas en [1] y en [2]. Ya que la filosofía que se va a usar es la misma que en los otros proyectos al igual que las herramientas y entorno de programación.

Concretando, los objetivos de este trabajo fin de grado son dos:

- Diseño e implementación de una interfaz gráfica realizada en Java que represente las simulaciones y estadísticas correspondientes.
- Realización de una aplicación que ejecute los cálculos necesarios para las simulaciones, en un CAS (sistema algebraico computacional), en nuestro caso Maxima.

Maxima es un CAS cuyo objetivo es la realización de cálculos matemáticos, tanto simbólicos como numéricos que es capaz de manipular expresiones algebraicas y matriciales, derivar e integrar funciones, realizar diversos tipos de gráficos, etc. Los motivos de su elección son los siguientes:

- Es un software de código abierto.
- Programar en Maxima es más amigable que en la mayoría de CAS.
- Existen herramientas para la comunicación Java-Maxima.

Java es un lenguaje de programación desarrollado por Sun Microsystems, orientado a objetos y sus principales características son: herencia simple, polimorfismo de datos, redefinición de métodos y vinculación dinámica. Lo hemos elegido por:

- Es un lenguaje multiplataforma.
- Soporta comunicaciones entre procesos.
- Útil y sencillo a la hora de realizar interfaces gráficas de usuario.

1.3 Metodología

La metodología de desarrollo que hemos decidido seguir es la técnica de prototipado simple o evolutiva, la cual consiste en plantear objetivos parciales a través de las distintas fases del proyecto. De esta manera no se define desde el principio todo el proyecto, sino que se ofrece un prototipo desde la primera etapa del mismo y a medida que se van completando las fases parciales, llamadas versiones, éstas se irán cargando de funcionalidad e irán reemplazando a la que les precedían hasta el final del proyecto.

A pesar de que los objetivos del proyecto fueron definidos desde el principio proporcionando una visión de todo el conjunto del proyecto, la elección de esta metodología nos ha proporcionado la flexibilidad que queríamos para el desarrollo del mismo, ya que las decisiones referentes al diseño de la interfaz y la programación no fueron definidas al comienzo

1.4 Estructura de la memoria

En el capítulo 1 de esta memoria se pretende introducir al lector el problema que pretendemos abordar y como pretendemos hacerlo. Se divide en cuatro secciones en las que se ha explicado la motivación del trabajo, los objetivos, la metodología con la que se ha llevado a cabo y, por último un resumen de cómo está estructurada el resto de esta memoria.

En el capítulo 2 se hace una descripción general de los aspectos y decisiones fundamentales de este trabajo y en el 3 se describe como se ha modelado el sistema y la justificación de porque se ha realizado así.

En el capítulo 4 se ha explicado la comunicación que lleva a cabo Java con Maxima y en el 5 se ha realizado un resumen del funcionamiento de la simulación, es decir, de todo el cálculo que se realiza en Maxima.

Seguidamente en los capítulos 6 y 7 se habla de tareas que se realizan en Java, la representación gráfica y el tratamiento de posibles errores. En el capítulo 8 podemos ver una manual de usuario de la aplicación.

Para concluir encontramos las conclusiones a las que hemos llegado tras la realización de este trabajo, referencias a la bibliografía que se ha utilizado y anexos.

2 Descripción del trabajo

2.1 ¿Por qué usamos un CAS?

El por qué usamos un CAS (siglas de Computer Algebra System, que significa sistema automático de cálculo simbólico) es debido a la necesidad de disponer de una herramienta matemática con la suficiente potencia como para realizar cálculos simbólicos.
En matemáticas y ciencias de la computación el cálculo simbólico o cálculo algebraico, es un área científica que se refiere al estudio y desarrollo de algoritmos y software para la manipulación de expresiones matemáticas y otros objetos matemáticos.

Los cálculos realizados con el CAS serán necesarios para la generación de personas en las distintas paradas del sistema y para alterar los tiempos entre una parada y otra, en función de la cantidad de tráfico deseada.

Otra razón para utilizar un CAS en nuestra aplicación es la necesidad de realizar algunos cálculos implícitos y utilizar en ciertas ocasiones números "enormes".

2.2 Funciones de distribución

En este sistema usamos por defecto 3 funciones de distribución, la exponencial, la de Poisson y la normal.

Las dos primeras las usamos para la generación de personas, usamos las dos juntas para ello debido a que tienen una relación muy estrecha, a pesar de que una se trata de una función de distribución continua y la otra discreta.

El proceso poissoniano plantea la ocurrencia de un valor de la variable de tipo poisson, en un entorno de tiempo, de área, espacio, etc, y se define para ello como “el número de veces (éxitos) que ocurre un cierto evento en dicho entorno”. Por ejemplo, número de llamadas telefónicas en un intervalo de tiempo, número de clientes de una tienda que son atendidos por hora, número de vehículos que llegan a una gasolinera cada 5 minutos, etc.

Por otro la distribución exponencial pretende explicar el de la medida por cada ocurrencia del evento (cada vez que ocurre éxito). Por ejemplo, el tiempo transcurrido entre dos llamadas telefónicas sucesivas, el tiempo transcurrido entre llegadas sucesivas de dos clientes a una tienda, longitud de tiempo transcurrido entre dos vehículos sucesivos que llegan a una gasolinera, etc.

Vemos claramente una identificación en el mismo modelo, el comportamiento de una variable poissoniana y una variable exponencial, mientras que la variable de poisson describe el número de ocurrencias de un

cierto evento por unidad de medida, la variable exponencial describe el valor de la medida entre dos ocurrencias sucesivas de dicho evento.

En el caso particular de esta aplicación, la variable exponencial determina cada cuanto tiempo llega gente a las paradas y la variable de poisson determina la cantidad de gente que llega.

La función de distribución normal es la función por defecto encargada en este sistema de modificar la duración de los tramos para así lograr una mayor aproximación a la realidad. Para cada tramo calculamos su nueva duración mediante una distribución normal con media el tiempo establecido para ese tramo y una cierta desviación que dependerá de la cantidad de tráfico que pretendamos que exista en la simulación.

En el siguiente código podemos ver como son en Maxima las funciones que acabamos de indicar:

```
aleatorio(a,b):=block(
       a+(b-a)*random(1.0)
)$


exponencial(lambda):=block(
       (-1/lambda)*log(aleatorio(0.0,1.0))
)$


poisson(lambda):=block(
       [c,b,k,aux],
       c:(%e)^(-lambda),
       b:c,
       k:0,
       aux:aleatorio(0.0,0.1),
       while aux>b do(k:k+1,c:(lambda/k)*c,b:b+c),k
)$


normalG(media,varianza) := block(
media + (varianza^0.5)*(((-
2*log(uniformeG(0.0,1.0)))^0.5)*cos(2*%pi*uniformeG(0.0,1.0)))
)$


uniformeG(a,b) := block(
a + (b-a)*aleatorio(0.0,1.0)
)$
```

Tabla 1. Código funciones distribución Maxima

Aunque estas son las funciones implementadas por defecto en la aplicación existe la posibilidad de cambiarlas mediante [8], con esto se logra un mayor abanico de posibles pruebas pues se obtendrán diferentes resultados al usar unas funciones de distribución u otras.

2.3 Utilidad Java para visualizar una matriz

Para realizar este trabajo se ha realizado un pequeño programa en el que poder modelar un mapa fácilmente a partir de una matriz de números enteros. Este programa nos será muy útil a la hora de crear el entorno gráfico y modelar el mapa principal que usará la aplicación.

El código de dicho programa en JAVA es el siguiente:

```
/**
 *  Primera versión del programa para visualizar mapas
 *  leyendo los datos de una matriz de enteros
 */

import java.awt.*;
import javax.swing.*;

public class dibujaMatrices extends JFrame {

	// Matriz que contiene el mapa
	public int Mapa[][] = {
			{ 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1 },
			{ 1, 1, 2, 2, 2, 2, 2, 1, 1, 2, 2, 2, 2, 2, 1, 1, 1 },
			{ 1, 1, 2, 1, 1, 1, 2, 2, 2, 2, 1, 1, 1, 2, 2, 2, 1 },
			{ 1, 1, 2, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 1 },
			{ 1, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 1 },
			{ 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1 } };

	// constructor con barra de título y dimensiones de la ventana
	public dibujaMatrices() {

		// título:
		super(" M A P A ");
		// tamaño más un margen por el borde:
		setSize(Mapa[1].length * 25 + 20, Mapa.length * 25 + 25 + 20);
		// Hacer Visible:
		setVisible(true);

		Container contenedor;
		contenedor = getContentPane();
		// Color de fondo (margen exterior):
		contenedor.setBackground(Color.BLUE);
	}

	// dibujar cada una de las zonas y objetos en distintos colores
	public void paint(Graphics g) {
		// llamar al método paint de la superclase
		super.paint(g);

		for (int i = 0; i < Mapa.length; i++) {
			for (int j = 0; j < Mapa[i].length; j++) {
```

```
                        // establecer el color de cada terreno
                        switch (Mapa[i][j]) {
                        case 1:
                              g.setColor(Color.BLACK);
                              break;
                        case 2:
                              g.setColor(Color.GRAY);
                              break;
                        default:
                              g.setColor(Color.ORANGE);
                              break;
                        }
                        g.fillRect(10 + j * 25, 10 + (i + 1) * 25, 25, 25);
                  }
            }

      } // fin del método paint

      // ejecutar la aplicación
      public static void main(String args[]) {
            dibujaMatrices aplicacion = new dibujaMatrices();
            // activar X para salir:
            aplicacion.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
            aplicacion.setBackground(Color.BLUE);
      }
} // fin
```

Tabla 2. Código Java del visualizador de matrices

El resultado obtenido al ejecutar esta aplicación es el siguiente:

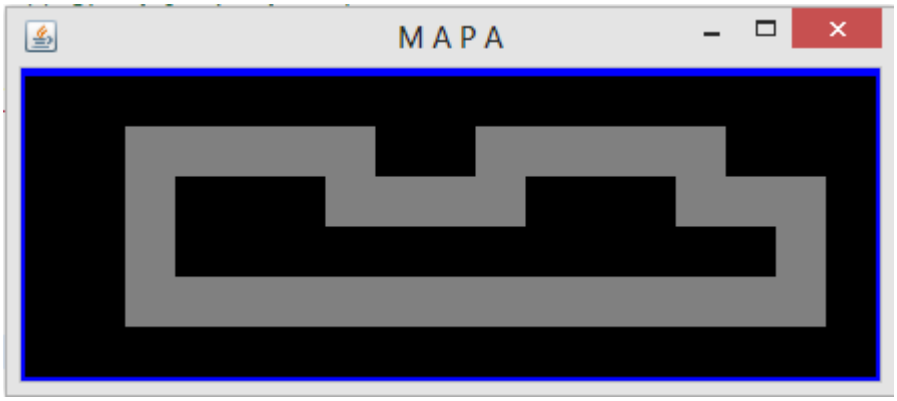


Imagen 1. Ejemplo de mapa

2.4 Creación del mapa

Tal y como hemos visto en el ejemplo anterior el mapa va a ser una matriz de enteros, donde cada número representará un tipo de terreno distinto.

Uno de los objetivos que se pretendía que tuviese este simulador era que no fuese estático, es decir que no sólo se pudiesen hacer cálculos sobre un mapa predefinido. Por ejemplo en [2], los cálculos solamente se pueden realizar sobre un mapa seleccionado por el programador, para el usuario poder cambiar el mapa es necesario que tenga conocimientos sobre programación para cambiar el código fuente, y esto limita mucho el tipo de usuario que puede

usar ese simulador. Es por esto que en este simulador cada vez que se inicia se permite que el usuario introduzca los datos que desee para así poder trabajar sobre ellos.

Los datos necesarios para la creación del mapa son las coordenadas de todas las paradas de cada línea, coordenadas X e Y. Para no poner límites ni dificultades al usuario en ningún momento se pide el tamaño máximo del mapa, en su lugar se examinan todas las coordenadas introducidas y se crea una matriz tal que permita albergar un mapa con todas las coordenadas introducidas.

Dadas las coordenadas de una parada, la parada A, y las coordenadas de la parada siguiente, la parada B, la forma de unir en el mapa ambas paradas es mediante el siguiente algoritmo:

```
      xA // Coordenada X paradaA
      yA // Coordenada Y paradaA
      xB // Coordenada X paradaB
      yB // Coordenada Y paradaB

    // Unimos horizontalmente
    if (xA < xB) {
    // Caso X-A esta a la izquierda de X-B
            while (xA < xB) {
            // Dibujamos hacia la derecha
                  xA = xA + 1;
                  Mapa[xA][yA]=indicadorDeLinea;
                        caminoAux.add(yA);
                        caminoAux.add(xA);
                  }
      }else {
      // Caso X-A esta a la derecha de X-B
            while (xA > xB) {
            // Dibujamos hacia la izquierda
                  xA = xA - 1;
                  Mapa[xA][yA]=indicadorDeLinea;
                        caminoAux.add(yA);
                        caminoAux.add(xA);
                  }
      }
       // Unimos verticalmente
      if (yA < yB) {
        // Caso Y-A esta debajo de Y-B
                  while (yA < yB) {
                  // Dibujamos hacia arriba
                        yA = yA + 1;
                        Mapa[xA][yA]=indicadorDeLinea;
                        caminoAux.add(yA);
                        caminoAux.add(xA);
                  }
      }else{
       // Caso Y-A esta arriba de Y-B
            while (yA > yB) {
             // Dibujamos hacia abajo
                        yA = yA - 1;
                        Mapa[xA][yA]=indicadorDeLinea;
                        caminoAux.add(yA);
                        caminoAux.add(xA);
            }
      }
```

Tabla 3. Código Java de unión de dos paradas

Para lograr una mayor diversidad a la hora de crear los mapas, se ha hecho que se vayan alternando dos formas de unir las paradas, es decir, mientras unas veces se comienza uniendo horizontalmente y luego verticalmente, como en el código anterior, otras veces se une en primer lugar verticalmente y luego horizontalmente.

Se puede ver en el código anterior que al mismo tiempo que se va creando el mapa se van almacenando las nuevas coordenadas en una estructura aparte, la lista de número enteros caminoAux, que posteriormente servirá para crear la matriz Caminos. El almacenamiento de estas coordenadas que unen una parada con otra es necesario para que los autobuses y el metro sepan que camino tienen que seguir para poder realizar la línea que tengan asignada.

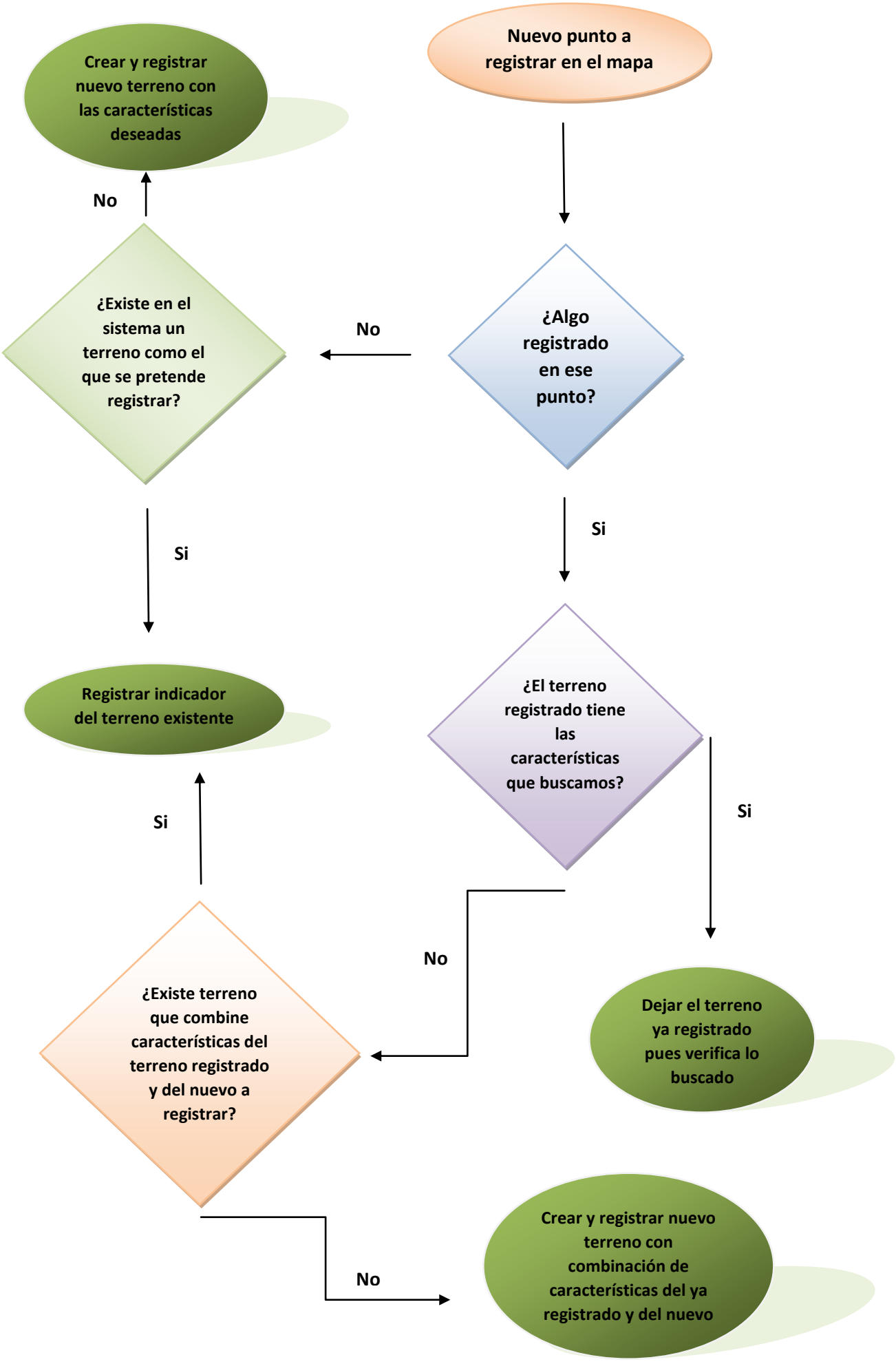
La variable identificador de terreno que se va indicando en la matriz Mapa dependerá del tipo de terreno que se esté registrando. La cantidad de terrenos distintos que pueden llegar a existir es otro aspecto a tratar. Por ejemplo en [2], se sabía que existían 7 tipos de terreno distintos, ni más ni menos, y aunque en este sistema sólo van a existir dos grandes categorías distintas de terrenos, carreteras o paradas, dentro de estas categorías el número de terrenos dependerá del tamaño del mapa. Como el mapa dependerá de los datos introducidos por el usuario la creación de los distintos terrenos a representar en el mapa se realizará al mismo tiempo que se va creando el mapa.

Cada vez que se quiera registrar algo en el mapa se podrán dar dos casos:

- Que no haya nada registrado, entonces se buscará si ya existe en el sistema un tipo de terreno como el que pretendemos indicar en el mapa, si existe entonces se indicará el identificador de ese terreno, en caso de que no exista, se creará un nuevo terreno con las características deseadas.

- Que ya haya algún terreno registrado, en ese caso en primer lugar se comprobara si el terreno registrado dispone de las características del terreno que pretendemos registrar, en ese caso no se registrara nada pues ya hay registrado algo como lo que deseamos. En el caso de que el terreno que se encuentra registrado no tenga las características de lo que queremos registrar, se procederá a buscar si existe registrado en el sistema algún terreno que combine las características que deseamos y las características ya registradas en ese punto, en el caso de que exista, se indicará el identificador de ese terreno, si no existe se creará un nuevo terreno que combine ambas características.

En la siguiente imagen se puede ver un diagrama de flujo que representa el algoritmo que acabamos de describir.


Nuevo punto a registrar en el mapa
¿Algo registrado en ese punto?
No
¿Existe en el sistema un terreno como el que se pretende registrar?
No
Crear y registrar nuevo terreno con las características deseadas
Si
¿El terreno registrado tiene las características que buscamos?
Si
Dejar el terreno ya registrado pues verifica lo buscado
No
¿Existe terreno que combine características del terreno registrado y del nuevo a registrar?
Si
Registrar indicador del terreno existente
No
Crear y registrar nuevo terreno con combinación de características del ya registrado y del nuevo

2.5 Funcionamiento

El funcionamiento del sistema tendrá tres protagonistas:

- Usuario
- Java (GUI)
- Maxima (CAS)

Primero el usuario interactuará con la interfaz gráfica (GUI) implementada en Java y durante la simulación cada vez que se produce un “paso”, Java a través de su librería jacomax llamará a Maxima (CAS) que se encargará de realizar todos los cálculos necesarios, quien posteriormente los devolverá a Java que por último mostrará la actualización de los datos por pantalla como podemos observar en la siguiente figura:

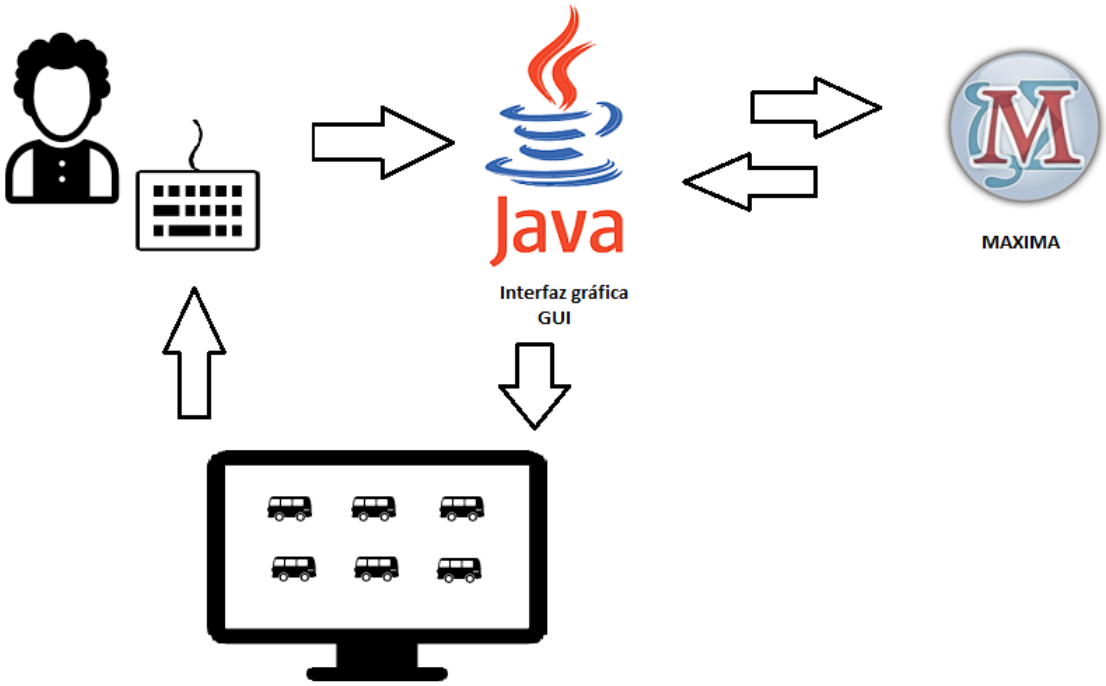


Imagen 2. Esquema de funcionamiento

3 Modelado del sistema

3.1 Representación de la información

La primera decisión a tomar, una vez adquirida la base teórica sobre la que se desarrollará la aplicación, es la forma en la que vamos a modelar el sistema, es decir, definir una estructura abstracta que documente y organice la información necesaria para simular nuestro sistema. Así como la forma en la que se almacenarán los datos y como se accederán a ellos.

Hay que tener en cuenta que aunque toda la información se va a guardar en Java, también se va a pasar a Maxima para que éste realice los cálculos necesarios para avanzar. Por tanto de primera mano se ha descartado la posibilidad de hacer una declaración de clases en Java ya que todas ellas habría que transformarlas en un tipo de datos que acepte Maxima. Por ello la decisión final ha sido la de implementar con arrays y matrices de estado el sistema que almacenamos en Java.

Las principales estructuras necesarias para este fin son las siguientes:

1) Mapa: Matriz que contiene los datos del terreno.
2) Terrenos: Matriz con información de los distintos tipos de terreno que existen.
3) Líneas: Matriz que con tiene las líneas de autobuses y metro y sus principales características.
4) Autobuses: Matriz que contiene los autobuses y vagones de metro del sistema.
5) Paradas: Matriz con información concerniente a las paradas de autobuses o metro.
6) Personas: Matriz con información sobre las personas existentes en el sistema.
7) Tramos: Matriz con datos sobre los distintos tramos existentes.
8) Hora: Matriz que contiene todas las horas necesarias para el correcto funcionamiento del sistema.
9) Caminos: Matriz con las coordenadas de los caminos de cada autobús o metro.
10) Estadísticas: Matriz encargada de recoger datos estadísticos.
11) Tráfico: Array con datos sobre los distintos tipos de cantidad de tráfico que admite el sistema.

Además de estas estructuras han sido necesarias otras estructuras auxiliares, constantes, contadores y otras variables para usos específicos propias de cada subsistema que se ha implementado. Veamos en más detalle las estructuras anteriormente indicadas.

3.2 Representación del mapa

La representación del mapa se lleva a cabo usando dos matrices: Mapa y Terrenos. La primera matriz es una matriz de enteros que contiene la información necesaria para la representación gráfica del mapa. El tamaño de esta matriz dependerá de los datos introducidos.

Cuando el número que aparece es un 0 significa que en ese punto del mapa no hay nada registrado, y por tanto a la hora de su representación gráfica se coloreará ese punto con el color establecido para ese tipo de zonas.

Para todo número *k* que aparezca en la matriz, siendo *k* mayor o igual que 1 es necesario buscar en la matriz Terrenos el tipo de terreno del que se trata, las características del terreno en cuestión vendrán indicadas en la fila *k* de la matriz.

En este sistema sólo existen dos grandes categorías de terrenos: paradas o carreteras. Dentro de la categoría paradas una parada será diferente de otra cuando tengan distinta ubicación, es decir, distintas coordenadas. En la categoría carreteras consideraremos una carretera distinta de otra cuando por cada una de ellas no pasen exactamente las mismas líneas.

Por tanto para cada terreno se guardaran los siguientes datos en la matriz Terrenos:

- [N][0]: Número que indica la categoría del terreno:
  -1. Indica que esta fila de la matriz no está siendo usada.
  0. Indica que el terreno se trata de una carretera.
  1. Indica que el terreno se trata de una parada.
- [N][i]: Número que indica las características del terreno en cuestión:
  0. Indica que la línea con identificador i no pasa por este terreno.
  1. Indica que la línea con identificador i pasa por este terreno.
  2. Indica que la línea con identificador i para en este terreno pues se trata de una parada.

La matriz Terrenos tendrá un tamaño de Nxl , donde N es el máximo número de combinaciones posibles e I es el número de líneas del sistema, distinguiendo entre líneas de ida y líneas de vuelta.

3.3 Matriz líneas

En este sistema podemos ver intuitivamente las líneas de autobuses o metro como una secuencia ordenada de tramos, donde los tramos sirven para unir un par de paradas distintas. La matriz líneas contiene los datos

relacionados a las líneas de estos transportes, de tamaño NxM donde N corresponde al número de líneas del sistema, tanto líneas de ida como líneas de vuelta, y M es igual a 10 + Tr, donde Tr es el número de tramos que contiene la línea con más tramos del sistema. Para cada línea se guardaran los siguientes datos:

- [N][0]: Número que indica el sentido de la línea, 0: sentido ida, 1: sentido vuelta.
- [N][1]: Identificador de la línea.
- [N][2]: Número de paradas de la línea.
- [N][3]: Identificador de la línea en sentido contrario a esta.
- [N][4]: Número de identificación de la línea, común a la línea de ida y a la de vuelta.
- [N][5]: Número de autobuses circulando en esta línea.
- [N][6]: Código del color de la línea para su representación gráfica.
- [N][7]: Posición (fila de la matriz) donde comienzan las coordenadas de esta línea en la matriz camino.
- [N][8]: Frecuencia con la que pasan los autobuses, en minutos.
- [N][9]: Identificador del primer tramo de la línea.
- [N][10]: Identificador del segundo tramo de la línea.
- …
- [N][M]: Variable auxiliar para Maxima que indica con un 0 que no tiene que salir un autobús o vagón de metro y con un 1 que es la hora de comience a circular un autobús o vagón de metro.

3.4 Matriz autobuses

Matriz que contiene los datos concernientes al metro y los autobuses que circulan por nuestro sistema o que están en alguno de los garajes esperando a que sea necesaria su entrada. De tamaño Nx16 donde N indica el número de vagones de metro y autobuses en el sistema y el propio identificador de cada vagón de metro y autobús. Para cada uno se incluirán los siguientes datos:

- [N][0]: Coordenada Y en la que se encuentra el autobús o metro.
- [N][1]: Coordenada X en la que se encuentra el autobús o metro.
- [N][2]: Capacidad máxima de pasajeros que admite el autobús o metro.
- [N][3]: Cantidad de pasajeros subidos en el autobús o metro.
- [N][4]: Número de la línea a la que pertenece.
- [N][5]: Sentido que se encuentra realizando, 0: ida, 1: vuelta.
- [N][6]: Cantidad de pasajeros que se han subido en la última parada por la que pasado.
- [N][7]: Número que indica el estado del autobús o metro:
  -1. Nada registrado, fila disponible para registrar un nuevo autobús o vagón de metro.
  0. Autobús en garaje.
  1. Autobús circulando.

K. Número mayor que uno que indica el tiempo (segundos) que tiene que estar parado el autobús en una parada mientras suben y/o bajan pasajeros.

- [N][8]: Posición (fila) de la matriz caminos en la que se encuentran sus coordenadas actuales.
- [N][9]: Indica posición (fila) de la matriz caminos donde comienza su camino de ida.
- [N][10]: Indica posición (fila) de la matriz caminos donde comienza su camino de vuelta.
- [N][11]: Cantidad de pasajeros que se han bajado en la última parada por la que ha pasado.
- [N][12]: Cantidad de tiempo (segundos) que tiene que estar en cada posición del tramo actual en el que se encuentra.
- [N][13]: Cantidad de tiempo (segundos) restante para realizar el siguiente movimiento.
- [N][14]: Variable auxiliar que dependiendo de su valor Maxima realiza un cálculo u otro.
- [N][15]: Número que indica su turno de espera cuando está parado en una parada esperando su hora de salir.

3.5 Matriz paradas

Matriz con información sobre las paradas de autobuses o metro existentes en el sistema. De tamaño Nx9 donde N indica el número de paradas. Para cada una se incluyen los siguientes datos:

- [N][0]: Identificador de la parada.
- [N][1]: Coordenada X de la parada.
- [N][2]: Coordenada Y de la parada.
- [N][3]: Segundos que faltan para que se vuelvan a generar personas en esta parada.
- [N][4]: Cantidad de gente esperando en la parada.
- [N][5]: Número de líneas que pasan por esa parada.
- [N][6]: 0: Indica que no hay ningún autobús ni metro parado en la parada, 1: indica que hay un autobús o metro parado en la parada.
- [N][7]: Parámetro de la función de distribución que indica cada cuanto tiempo llega gente a la parada.
- [N][8]: Parámetro de la función de distribución que determina cuanta gente llega a la parada.

3.6 Matriz personas

Matriz que contiene la información de las personas existentes en el sistema, de tamaño Nx3, donde N es el número de personas en el sistema y el identificador de cada persona. Para cada persona se disponen de los siguientes datos:

- [N][0]: Número que indica el estado de esta persona:
  -2. Indica que la persona representada en esta fila ya ha finalizado su recorrido y sus datos están guardados hasta que sean registrados las estadísticas.
  -1. Indica que esta fila se encuentra libre para la creación de una nueva persona.
  0. Indica que la persona se encuentra esperando en una parada.
  1. Indica que la persona se encuentra viajando en un autobús o metro.
- [N][1]: Identificador de la parada a la que se dirige, parada destino.
- [N][2]: Si la persona se encuentra esperando en una parada indica el identificador de la parada en la que espera. Si la persona se encuentra viajando en un autobús o metro, indica el identificador del autobús o metro.

3.7 Matriz tramos

En esta matriz se almacenan los datos correspondientes a los tramos. De tamaño Nx6, donde N indica el número de tramos. Se disponen de los siguientes datos sobre los tramos:

- [N][0]: Identificador de la para inicial del tramo, parada A.
- [N][1]: Identificador de la parada final del tramo, parada B.
- [N][2]: Hora de salida de la parada A (formato hh:mm).
- [N][3]: Hora de salida de la parada B (formato hh:mm).
- [N][4]: Tiempo entre paradas A y B (minutos).
- [N][5]: Identificador del tramo.

3.8 Matriz hora

Matriz con información detallada de todas las horas que se usan en el sistema. De tamaño Nx8. Se disponen de los siguientes datos:

- [N][0]: Identificador del tramo del que se está guardando información.
- [N][1]: Campo del tramo al que pertenece:
  0. Parada A.
  1. Parada B.
  2. Duración A-B.
- [N][2]: Decenas de las horas.

- [N][3]: Unidades de las horas.
- [N][4]: Decenas de los minutos.
- [N][5]: Unidades de los minutos.
- [N][6]: Decenas de los segundos.
- [N][7]: Unidades de los segundos.

En la fila 0 de esta matriz se indica cuanto tiempo aumenta en cada paso que nos comunicamos con Maxima, este tiempo viene indicado en segundos. Este tiempo también aparece descompuesto en la última fila de esta matriz.

En la penúltima fila de esta matriz se encuentra almacenada de forma descompuesta la hora actual del sistema.

En la antepenúltima fila guardamos de forma descompuesta el tiempo de subida/bajada de las personas en el autobús.

En la posición anterior a la antepenúltima fila guardamos el aumento que se produce en cada comunicación que tenemos con Maxima, pero esta vez multiplicado por 2.

En la posición antepenúltima menos 2 se encuentra almacenada la hora actual más el incremento multiplicado por 2.

3.9 Matriz caminos

Matriz con los datos de todos los posibles caminos a seguir por los autobuses y metros en el sistema. De tamaño Nx2, donde N dependerá de las líneas introducidas. Cada vez que acaba un camino se indica insertando una fila con las coordenadas (-1,-1). Los datos almacenados son:

- [N][0]: Coordenada X del camino.
- [N][1]: Coordenada Y del camino.

3.10 Matrices estadísticas

Dentro de las matrices estadísticas se distinguen 4 tipos de matrices estadísticas que son:

3.10.1 Matriz estadísticas personas

En esta matriz se recogen todos los datos estadísticos asociados a las personas que se encuentran en el sistema. Esta matriz tiene un tamaño de Nx11, donde N es el número de personas en el sistema. La persona con el identificador N tiene guardados sus datos estadísticos en la fila N de esta matriz. Los datos que se guardan son:

- [N][0]: Número de la línea que ha usado la persona.
- [N][1]: Identificador de la parada que fue el origen de la persona.

- [N][2]: Identificador de la parada que fue el destino de la persona.
- [N][3]: Hora de llegada de la persona a su parada origen.
- [N][4]: Minuto de llegada de la persona a su parada origen.
- [N][5]: Hora de subida de la persona al autobús.
- [N][6]: Minuto de subida de la persona al autobús.
- [N][7]: Hora de llegada de la persona a su destino.
- [N][9]: Minutos de llegada de la persona a su destino.
- [N][10]: Identificador del autobús o metro usado por la persona.
- [N][11]: Número de autobuses que ha tenido que esperar la persona hasta poder subir al autobús o metro.

3.10.2 Matriz estadísticas autobús ida y matriz estadísticas autobús vuelta

La matriz estadísticas autobús ida y la matriz estadísticas autobús vuelta tienen la misma estructura. Se han creado 2 matrices para que en una registren los datos los autobuses y vagones de metro que finalizan su camino de ida y en la otra los que finalizan su camino de vuelta, para así facilitar los cálculos posteriores y evitar posibles pérdidas de datos. El tamaño de estas matrices es NxM, donde N es el número de autobuses y metros registrados en el sistema y M es igual a 12 más el número máximo de paradas que tenga una línea, multiplicado por 4. Los datos que se registran en esta matriz son:

- [N][0]: Identificador del autobús o metro que ha realizado la línea en cuestión.
- [N][1]: Número de la línea.
- [N][2]: Identificador de la parada inicial.
- [N][3]: Identificador de la parada final.
- [N][4]: Hora fijada de salida desde la parada inicial.
- [N][5]: Minuto fijado de salida desde la parada inicial.
- [N][6]: Hora real de salida desde la parada inicial.
- [N][7]: Minuto real de salida desde la parada inicial.
- [N][8]: Hora fijada de llegada a la parada destino.
- [N][9]: Minuto fijado de llegada a la parada destino.
- [N][10]: Hora real de llegada a la parada destino.
- [N][11]: Minuto real de llegada a la parada destino.

A partir de esta fila todas las demás columnas van en grupos de 4, un grupo por cada parada en la que para el autobús o metro. Estos grupos de 4 columnas recogen la siguiente información:

- [N][M]: Hora de llegada a la parada.
- [N][M+1]: Minuto de llegada a la parada.
- [N][M+2]: Cantidad de gente que se baja del autobús en esa parada.
- [N][M+3]: Cantidad de gente que se sube en el autobús en esa parada.

3.10.3 Matriz estadísticas de autobuses perdidos

En esta matriz se recogen los datos de aquellos autobuses o metros que han dejado de realizar una expedición. El tamaño de esta matriz es de Nx4, donde N es el número de autobuses en el sistema. Los datos recogidos en esta matriz son:

- [N][0]: Identificador de la línea que no se ha realizado.
- [N][1]: Indica el sentido de la línea, 0: ida , 1:vuelta.
- [N][2]: Hora a la que tendría que haber salido y no lo hizo.
- [N][3]: Minuto en el que tendría que haber salido y no lo hizo.

3.11 Array tráfico

En este array se recogen los datos sobre los distintos tipos de tráfico que se pueden dar en el sistema. El tamaño de este array es de 11 posiciones donde cada una es:

- [0]: Variable que indica a Maxima si tiene que recalcular la duración de los tramos donde 0: no y 1: si.
- [1]: Parámetro para modificar los tramos cuando el tipo de tráfico seleccionado es ligero.
- [2]: Parámetro para modificar los tramos cuando el tipo de tráfico seleccionado es moderado.
- [3]: Parámetro para modificar los tramos cuando el tipo de tráfico seleccionado es intenso.
- [4]: Variable que indica el tipo de tráfico establecido en el sistema, donde: 1 es ligero, 2 es moderado y 3 es intenso.
- [5]: Parámetro para generar personas cuando tráfico es ligero.
- [6]: Parámetro para generar personas cuando tráfico es moderado.
- [7]: Parámetro para generar personas cuando tráfico es intenso.
- [8]: Parámetro para indicar intervalo de llegada de personas con tráfico ligero.
- [9]: Parámetro para indicar intervalo de llegada de personas con tráfico moderado.
- [10]: Parámetro para indicar intervalo de llegada de personas con tráfico intenso.
- [11]: Varianza de la función normal, función por defecto encargada de determinar el tiempo que dura una avería.

3.12 Array averías

En este array se recogen los datos encargados de caracterizar las averías que se pueden producir en los autobuses o vagones de metro del sistema. El tamaño de este array es de cuatro posiciones donde cada una es:

- [0]: Cantidad de segundos restante para que se produzca la primera avería.
- [1]: Parámetro de la función que determina cada cuanto tiempo se produce una avería.
- [2]: Variable que en caso de tener valor 1 indica que se ha producido un error con la función de distribución en Maxima.
- [3]: Cantidad de averías que se han producido desde que se inició el sistema.
- [4]: Minutos que dura una avería.

4 Comunicación Java – Maxima

Para realizar la comunicación entre Java (interfaz gráfica) y Maxima (motor de cálculo) se ha usado la librería de código abierto JACOMAX (Java Connector for Maxima) [3]. Esta librería nos permite crear un proceso de Maxima al que se le pueden pasar las llamadas que requiramos (en formato texto) y nos devuelve las salidas de Maxima (también en texto).

Para usar esta librería es necesario seguir los siguientes pasos:

- Desde nuestro proyecto Java importar los archivos: "jacomax-0.2.3.jar", "slf4j-api-1.7.2.jar" y "slf4j-simple-1.7.2.jar" donde los números que indican la versión de estos archivos pueden variar.
- Añadir en nuestra carpeta del proyecto el archivo: "jacomax.properties" con el contenido siguiente:

```
jacomax.maxima.path=C\:\\Program Files (x86)\\Maxima-
5.31.2\\bin\\maxima.bat
jacomax.maxima.charset=Cp1252
```

Tabla 4. Archivo: "jacomax.properties"

En caso de tener Maxima instalado en una carpeta distinta a la indicada en jacomax.maxima.path, cambiar la ruta por la ruta oportuna.

- Usar los procedimientos indicados en el siguiente ejemplo de uso para el cual obtendremos como resultado "6":

```
public void Ejemplo(){
  String llamada, salida = null;

  llamada = "4+2;";
  MaximaConfiguration configuration= jacomaxSimpleConfigurator.configure();
  MaximaProcessLauncher launcher=new MaximaProcessLauncher(configuration);
  MaximaInteractiveProcess process= launcher.launchInteractiveProcess();
  try{
       salida = process.executeCall(llamada);
       process.terminate();
       }catch (MaximaTimeoutException e){
             e.printStackTrace();
       }
       System.out.println( salida );
}
```

Tabla 5. Ejemplo de uso de Jacomax

5 Descripción del funcionamiento

En este capítulo vamos hacer una descripción general del funcionamiento que sigue este sistema al iniciarse y en cada iteración, es decir, en cada comunicación de Java con Maxima.

La aplicación comienza solicitando la información necesaria para la creación de la estructura sobre la que se va a realizar los cálculos. Una vez que se introducen todos los datos Java se encarga de crear e inicializar todas las estructuras necesarias para el correcto funcionamiento del sistema. En ese momento la aplicación se encuentra lista para comenzar a ejecutarse, para ello pulsaremos el botón Play. En la siguiente imagen podemos ver todos los métodos encargados de inicializar las estructuras.

```
iniciaColores();
iniciaDatosAutobuses();
iniciaDatosAverias();
iniciaDatosLineas(numLineas, true);
iniciaDatosMapa();
iniciaDatosParadas();
iniciaDatosPersonas();
iniciaDatosTrafico();
iniciaDatosTramos();
iniciaEstadisticas();
iniciaHora(1);
```

Tabla 6. Métodos inician sistema

Al pulsar el botón Play Java se pone en contacto con Maxima, le envía las estructuras necesarias y Maxima comienzan los cálculos.

El primer cálculo que realiza Maxima es actualizar la hora global del sistema, es decir, a la hora global pasada, sumarle el tik de reloj establecido en el sistema. Esto se hace para ver qué cosas son las que van a suceder en el intervalo de tiempo que existe desde la hora mandada desde Java y la hora incrementada.

Una vez actualizada la hora y correctamente registrada, Maxima realiza dos comprobaciones, en primer lugar comprueba si es necesario modificar la duración de los tramos debido a que se haya producido un cambio en el tipo de tráfico establecido en el sistema y en segundo lugar comprueba si en esta iteración se tiene que producir una avería.

A continuación Maxima inicia un bucle que recorre todas las paradas registradas. Para cada parada siguiendo los resultados obtenidos por la función de distribución asignada se generará una cantidad de personas u otra.

Finalizada la generación de personas se inicia otro bucle, esta vez el bucle recorre todas las líneas comprobando si en el intervalo de tiempo actual debería iniciarse el viaje de un autobús o metro. En el caso de que sí se

procederá a comprobar si hay algún autobús o metro esperando para salir en la parada inicial de la línea, si lo hay se le indicará que debe de salir. En caso de que no haya ningún autobús o metro esperando y la línea sea de sentido ida, se comprobará si hay autobuses o metros de esa línea disponibles en el garaje, es decir, que aún no han entrado al sistema, en el caso de que si los haya entrará uno al sistema y comenzará la expedición. Si tampoco hay ningún autobús o metro para esta línea en el garaje entonces se dejará indicado para la siguiente iteración que en cuanto llegue un autobús o metro a la parada de esta línea deberá salir inmediatamente, pues irá con retraso.

El motivo de que la opción de comprobar si hay autobuses o vagones de metro en el garaje sea sólo en las líneas con sentido de ida es porque suponemos en el sistema que los autobuses y el metro siempre comienzan en la ida y no en la vuelta, es decir, para que un autobús realice una vuelta primero debe de haber realizado una ida.

El último bucle con el que se continúa el cálculo es un bucle que recorre todos los autobuses y vagones de metro dentro del sistema, este bucle tiene como objetivo mover a los autobuses y vagones de metro.

Una vez realizado este último bucle se devuelven todas las estructuras a Java y este se encarga de interpretar los resultados, representarlos gráficamente y construir las estadísticas. En el siguiente código podemos ver como enviamos todas las estructuras a Maxima y como Maxima las devuelve a Java para su representación.

```
public static void pasoMaxima(MaximaInteractiveProcess process) {

            // Llamar a MAXIMA
            String llamada, salida = null;

            llamada = Maxima.matrizToMaxima(Data.hora, "hora")
                        + Maxima.matrizToMaxima(Data.Caminos, "camino")
                        + Maxima.matrizToMaxima(Data.Autobuses, "buses")
                        + Maxima.matrizToMaxima(Data.Paradas, "paradas")
                        + Maxima.matrizToMaxima(Data.Personas, "personas")
                        + Maxima.matrizToMaxima(Data.Lineas, "lineas")
                        + Maxima.matrizToMaxima(Data.Terrenos, "terrenos")
                        + Maxima.matrizToMaxima(Data.MapaAux, "mapa")
                        + Maxima.matrizToMaxima(Data.Tramos, "tramos")
                        + Maxima.matrizToMaxima(Data.estadisticasPersonas,
                                    "estadisticas")
                        + Maxima.matrizToMaxima(Data.estadisticasBusIda,
"estadBusIda")
                        +
Maxima.matrizToMaxima(Data.estadisticasBusVuelta,
                                    "estadBusVuel")
                        +
Maxima.matrizToMaxima(Data.estadisticasExpedicionesPerdidas,
                                    "perdidas")
                        + Maxima.matrizToMaxima(Data.averias, "averias")
                        + Maxima.matrizToMaxima(Data.trafico, "trafico")
                        + "load(\"bus.mac\")$"+"hora;"+"camino;"+"buses;"
                        + "paradas;"+"personas;"+"lineas;"+"terrenos;"+
"mapa;"+ "tramos;" + "estadisticas;" + "estadBusIda;"
+ "estadBusVuel;" + "perdidas;" + "averias;" + "trafico;";
```

```
		try {
			process.softReset();
			salida = process.executeCall(llamada);
		} catch (MaximaTimeoutException e) {
			e.printStackTrace();
		}

		// segun el número de la salida deseada (%oi)
		int numero = 18;
		// guardamos la nueva matriz de hora obtenida
		maximaToMatriz(salida, Data.hora, numero);
		numero++;
		// guardamos la nueva matriz de caminos obtenida
		maximaToMatriz(salida, Data.Caminos, numero);
		numero++;
		// guardamos la nueva matriz de buses obtenida
		maximaToMatriz(salida, Data.Autobuses, numero);
		numero++;
		// guardamos la nueva matriz de paradas obtenida
		maximaToMatriz(salida, Data.Paradas, numero);
		numero++;
		// guardamos la nueva matriz de personas obtenida
		maximaToMatriz(salida, Data.Personas, numero);
		numero++;
		// guardamos la nueva matriz de lineas obtenida
		maximaToMatriz(salida, Data.Lineas, numero);
		numero++;
		// guardamos la nueva matriz de lineas obtenida
		maximaToMatriz(salida, Data.Terrenos, numero);
		numero++;
		// guardamos la nueva matriz de Mapa obtenida
		maximaToMatriz(salida, Data.MapaAux, numero);
		numero++;
		// guardamos la nueva matriz de tramos obtenida
		maximaToMatriz(salida, Data.Tramos, numero);
		numero++;
		// guardamos la nueva matriz de estadisticasPersonas obtenida
		maximaToMatriz(salida, Data.estadisticasPersonas, numero);
		numero++;
		// guardamos la nueva matriz de estadisticasLineas obtenida
		maximaToMatriz(salida, Data.estadisticasBusIda, numero);
		numero++;
		// guardamos la nueva matriz de estadisticasLineas obtenida
		maximaToMatriz(salida, Data.estadisticasBusVuelta, numero);
		numero++;
		// guardamos la nueva matriz de expediciones perdidas obtenida
		maximaToMatriz(salida, Data.estadisticasExpedicionesPerdidas,
numero);
		numero++;
		// guardamos la nueva matriz de averias
		maximaToMatriz(salida, Data.averias, numero);
		numero++;
```

Tabla 7. Código envía y recibe estructuras

5.1 Inicialización de la hora

Para inicializar la hora global del sistema se tomará la hora de la línea que comience más temprano, siendo la hora más temprana la hora más cercana a las 00:00. Una vez seleccionada la hora más temprana se procederá a restarle

3 veces el tik de reloj establecido, el restar 3 veces el tik de reloj se hace para que una vez que comience el funcionamiento de la primera línea ya haya “algo de vida” en el sistema, es decir, gente en las paradas. Por motivos de programación en Maxima todas las simulaciones que se realicen en esta aplicación deben realizarse entre las 00:00 y las 23:59, es decir, no pueden comenzar en un día y acabar en el siguiente.

5.2 Generación de personas

El bucle que recorre todas las paradas va comprobando que la columna 3 de la parada en cuestión tenga un número positivo. Si el valor que tiene es mayor que 0 lo único que se hace es restarle el tik de reloj establecido en el sistema. En el caso de que tenga un valor menor o igual que 0 significa que se deben generar personas en esa parada, es decir, llegan personas a la parada, el número de personas que llega lo determinamos por defecto mediante una Poisson.

Una vez determinado el número se inicia un nuevo bucle que se encarga de buscar en la matriz Personas si existen tantos huecos como se necesiten para registrar los datos de las nuevas personas. Un hueco es una fila donde en la columna 0 (espacio reservado para el identificador de la persona) aparece el número -1.

Una vez que se han localizado los huecos lo primero que se hace para cada persona es comprobar todos los posibles destinos que se pueden alcanzar desde la parada en la que nos encontramos. Si este número de destinos posibles es N, entonces se genera aleatoriamente un número entre 1 y N, este número aleatorio va a ser el destino de la persona en cuestión.

Una cosa más que se realiza cuando se generan las personas en el sistema es registrar en la matriz estadísticas personas la hora a la que ha llegado la persona a la parada, para posteriormente poder calcular el tiempo de espera hasta subirse en el autobús.

Finalmente cuando se han terminado de generar personas mediante una función de distribución exponencial se indica en la columna 3 de la parada el tiempo que tiene que volver a pasar para que se vuelvan a generar personas en la parada en cuestión.

5.3 Inicialización del metro y autobuses

Una vez que se han introducido todos los datos Java se encarga de generar un número determinado de autobuses y vagones de metro para cada línea introducida, este número determinado de autobuses y vagones de metro se muestra al usuario mediante una interfaz como la de la siguiente imagen y se da la opción de o bien aceptarlo o modificarlo a necesidad del usuario.

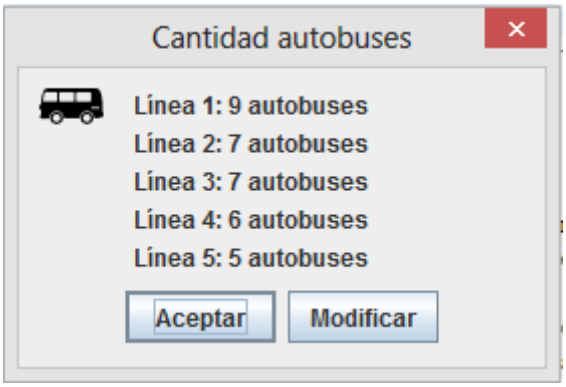


Imagen 3. Interfaz número de autobuses

El proceso que seguimos en Java para generar el número inicial de autobuses y vagones de metro es el siguiente:

Dada una línea, sumamos las duraciones de todos sus tramos de ida y de todos sus tramos de vuelta, además por cada parada que exista en cualquiera de los sentidos sumamos 30 segundos. Así obtenemos una cantidad de tiempo que dividimos por la frecuencia indicada para la línea en cuestión. El resultado de esta división es el número de autobuses que se generará para dicha línea.

5.4 Movimiento de los autobuses y metros

El movimiento de los autobuses y metros depende del tramo en el que se encuentre y del tik de reloj establecido en el sistema.

Cada vez que un autobús o metro llega al inicio de un tramo, es decir, a una parada, se procede a calcular cada cuantos segundos se tiene que mover el autobús o metro en el mapa. Para ello lo primero que se hace es contabilizar cuantas posiciones ocupa en el mapa el tramo en cuestión, después se consulta el tiempo de duración establecido para ese tramo y entonces con ambos datos se realiza una simple regla de 3 con la que obtenemos cuantos segundos tiene que pasar el autobús en cada posición de ese tramo. Este valor se registra en la columna 12 y 13 de la fila que contiene los datos del autobús en cuestión, en la matriz Autobuses.

Una vez realizado el cálculo anterior se procede a restar a la columna 13 el valor del tik de reloj establecido, en caso de que de cómo resultado un valor menor o igual que 0 habrá que realizar un movimiento, es decir, el autobús o metro tiene que ir a la siguiente posición que tiene asignada y volver a establecer en la columna 13 el valor indicado en la columna 12, a no ser que ya la siguiente posición sea una parada, es decir, el inicio de un nuevo tramo. Si el resultado de la resta es mayor que 0 es que aún no ha llegado el momento de que se mueva el autobús o metro.

Otro aspecto importante en el movimiento del metro y autobuses es que cada vez que un autobús o metro llega a una nueva posición comprueba si dicha posición se trata de una parada, y en ese caso si es una de las paradas

en las que tiene que parar dicho autobús o metro. Cuando todo esto ocurre, se comprueba si en esa parada hay gente esperando con un destino al que se dirige el autobús o metro en cuestión o si hay gente en el autobús o metro que desea bajarse en esa parada.

Tanto como cuando se sube una persona o se baja del metro o autobús registramos en la matriz estadísticas personas la hora a la que lo ha hecho.

Los autobuses y el metro saben el camino que tienen que seguir gracias a la matriz Caminos.

5.5 Averías

En la aplicación se ha dado la posibilidad de que los autobuses y el metro puedan sufrir averías en determinados momentos. La función de distribución por defecto encargada de determinar cada cuanto tiempo se produce una avería es la función de distribución exponencial. Cuando llega el momento en el que se tiene que producir una avería se selecciona aleatoriamente uno de los autobuses o vagones de metro que se encuentran circulando en el sistema. Una vez seleccionado el autobús que va a sufrir la avería el tiempo que dura la avería es por defecto de 15 minutos. La aplicación da la posibilidad de cambiar este tiempo.

La forma de diferenciar gráficamente un autobús o metro normal de uno averiado es por el color con el cual se representa, mientras que el autobús o metro normal se representa con un color rojo oscuro, el autobús o metro que sufre una avería se representa con el color negro.

5.6 Tráfico

Al igual que con las averías para lograr una mayor aproximación a la realidad se ha añadido al sistema la posibilidad de distinguir entre una cantidad de tráfico u otra. Se han creado 3 tipos de tráfico distintos:

- Ligero
- Moderado
- Intenso

La selección de un tipo de tráfico u otro influye en 3 cosas. En primer lugar en la forma de alterar el tiempo originalmente establecido en un tramo, en segundo lugar en la cantidad de personas que llegan a las paradas y en tercer lugar en cada cuanto tiempo llega gente a las paradas.

En caso de desear que alguno de estos tipos de tráfico se caracterice por una función distinta a las establecidas o cambiar los parámetros establecidos por defecto usaremos el módulo desarrollado en [8] que permite cambiar los paquetes de funciones que se cargan inicialmente.

5.7 Modificación de funciones por defecto

Como hemos indicado anteriormente las variables aleatorias que permiten ser modificadas son las que realizan la siguiente función:

- Determinación de cada cuanto tiempo llegan personas a una parada.
- Determinación de cuantas personas llegan a una parada.
- Determinación de la frecuencia en la que se producen averías en los autobuses.

Veamos un ejemplo de uso del programa realizado en [8] que crea un paquete Maxima de funciones de distribución en un proyecto de simulación de flujo de tráfico en tiempo acelerado.

Este programa crea siete paquetes Maxima, y se cargan en el fichero .mac que realiza la simulación, es decir, en el fichero de Maxima. Cada uno de ellos contiene una de las variables aleatorias antes definidas. Se diferencian los paquetes en función de los distintos tipos de tráfico. Estos paquetes son:

- frecuenciapersonasintenso.mac
- frecuenciapersonasmoderado.mac
- frecuenciapersonasligero.mac
- cantidadpersonasintenso.mac
- cantidadpersonasmoderado.mac
- cantidadpersonasligero.mac
- frecuenciaaverias.mac

Las variables aleatorias que tendrán por defecto cada uno de ellos en el mismo orden que han sido mostrados antes, son:

- exponencial de lambda 1
- exponencial de lambda 0.1
- exponencial de lambda 0.03
- poisson de lambda 5
- poisson de lambda 3
- poisson de lambda 1
- exponencial de lambda 0,003.

Cada vez que se pulsa en la opción encargada de modificar las funciones de distribución en cuestión se inicia [8]. Gracias a [8] podemos definir nuevas funciones o elegir alguna de las que ya tiene precargadas, en la siguiente imagen se puede ver la interfaz que permite que se seleccione una de las funciones ya definidas.

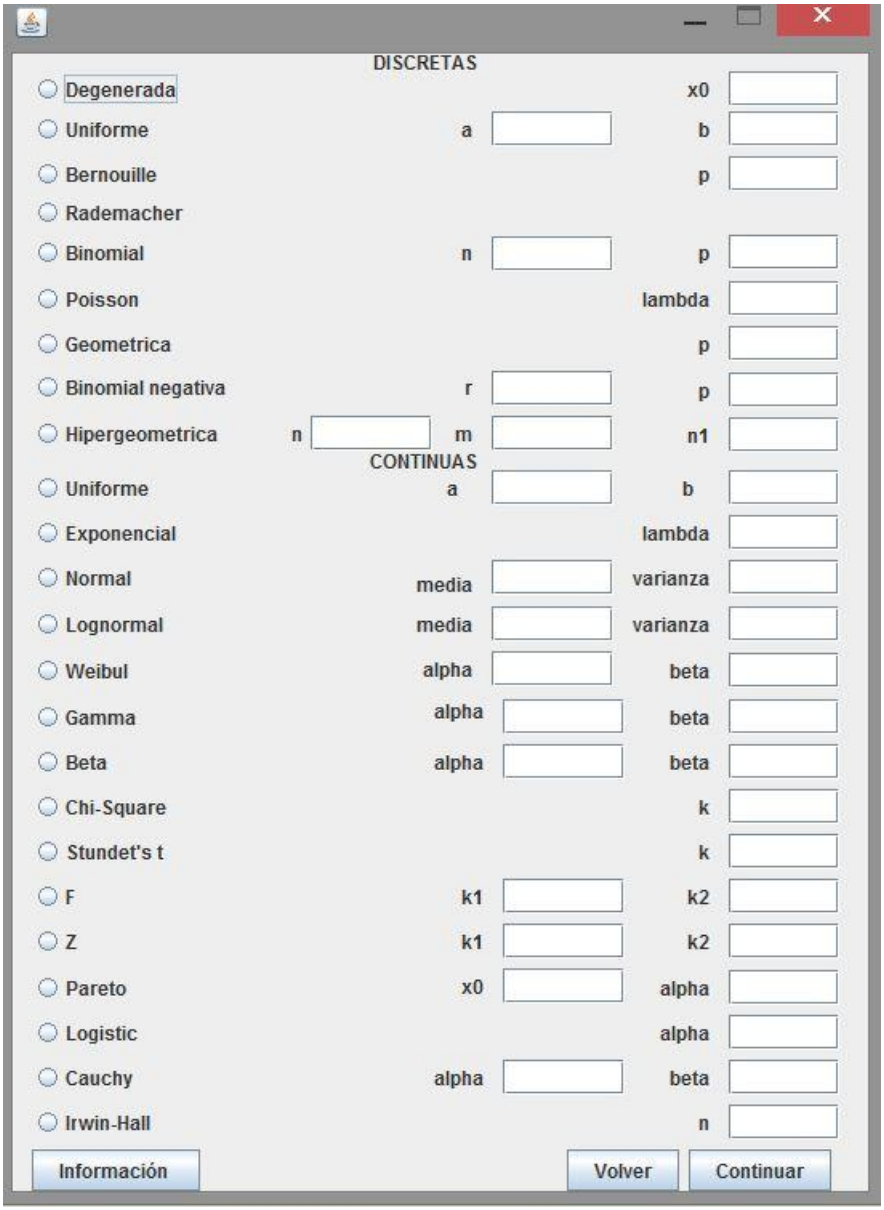


Imagen 4. Interfaz funciones [8]

En el fichero Maxima encargado de realizar la simulación del flujo de tráfico en nuestra aplicación, lo primero que se hace es cargar los paquetes que hemos indicado antes, atendiendo al tipo de tráfico que se encuentre establecido. En código será así:

```
load("C:/paquetesMaxima/frecuenciaaverias.mac")$

 /* Dependiendo del tipo de tráfico establecido se carga un
paquete u otro */

if (ceiling(trafico[5])=1)then(
   /* Tráfico ligero */
   load("C:/paquetesMaxima/frecuenciapersonasligero.mac"),
   load("C:/paquetesMaxima/cantidadpersonasligero.mac")
)else if (ceiling(trafico[5])=2)then(
   /* Tráfico moderado*/
```

```
        load("C:/paquetesMaxima/frecuenciapersonasmoderado.mac"),
        load("C:/paquetesMaxima/cantidadpersonasmoderado.mac")
      )else(
        load("C:/paquetesMaxima/frecuenciapersonasintenso.mac"),
        load("C:/paquetesMaxima/cantidadpersonasintenso.mac")
)$
```

Tabla 8. Código carga paquetes

Una vez realizados todos estos pasos, cada vez que en la simulación en Maxima se necesite un valor de una de las variables aleatorias basta con realizar una de las siguientes llamadas:

- frecuenciapersonasfG()
- cantidadpersonasfG()
- frecuenciaaveriasfG()

Se han realizado pequeñas modificaciones en el código del programa [8] para que se adapte a la nomenclatura que hemos seguido en esta aplicación.

La decisión de crear siete paquetes Maxima y cargar unos u otros dependiendo del trafico, en vez de crear un solo paquete, ha servido para que en cualquier momento se pueda cambiar la definición de una de las variables sin necesidad de modificar las otras o volverlas a introducir. Además el paquete se puede cambiar en cualquier momento, inclusive cuando el programa está en ejecución.

El dar la posibilidad de cambiar la función de distribución por cualquiera que desee el usuario hace que se puedan producir casos que no tengan sentido alguno y casos que conduzcan a errores. Una de las cosas que se ha realizado para evitar uno de los posibles errores ha sido comprobar cada vez que se genera una variable aleatoria si se trata de un número positivo, en caso de que no lo sea se genera un error avisando al usuario. El mensaje de error que se genera es como el de la siguiente imagen:

Imagen 5. Interfaz error función distribución

6 Representación gráfica de elementos

Java contiene los paquetes *java.awt* y *javax.swing* que junto con varios subpaquetes de éstos, son los encargados de proporcionar las clases necesarias para construir interfaces gráficas de usuario. A la hora de dibujar las carreteras, las paradas, los autobuses y los vagones de metro que circulan por el sistema, Java nos fue de gran utilidad, porque posee métodos para realizar dibujos sencillos, llamados a veces primitivas gráficas, además dispone de una clase llamada *Graphics*, que aparte de mantener un contexto gráfico formado por un área de dibujo actual, un color de dibujo del background (fondo) y otro del foreground (primer plano) en el que se realizarán las operaciones de dibujo, contiene los métodos para dibujar las primitivas gráficas, muy utilizadas en la elaboración de esta parte de la aplicación.

Así, por ejemplo, para dibujar las carreteras, es decir las líneas de autobuses y metro, y las paradas hemos utilizado el método *fillRect(int x,int y,int w,int h),* que dibuja un rectángulo en las coordenadas *x* e *y*, con una anchura de *w* y con una altura *h* y lo rellena con el color actual, que en este caso depende de la línea que se vaya a dibujar.

Los dos métodos que más hemos usado para implementar la parte gráfica de la aplicación han sido el método *paint()* y el método *repaint()*, métodos que pertenecen a la clase *Component* del paquete *java.awt.*

El método *paint()* es un método que no hace nada a no ser que no lo redefinamos en una de sus clase derivadas. Nosotros como programadores no tenemos que preocuparnos de llamar a este método, ya que el sistema operativo y Java lo llaman cada vez que entienden que la ventana debe ser dibujada o redibujada. El único argumento de *paint()* es un objeto g de la clase *Graphics*.

El método *repaint*() es el método que con más frecuencia hemos utilizado en el desarrollo de nuestra aplicación. Este método llama “lo antes posible” al método *update()* de la clase Component, que hace dos cosas: primero redibuja la ventana con el color de fondo y luego llama a *paint().*

Otra clase que hemos tenido muy en cuenta a la hora de dibujar las líneas de autobuses y metro además los propios autobuses y vagones de metro ha sido la clase *Color* contenida en el paquete *java.awt*, esta clase encapsula colores utilizando el formato RGB (Red, Green, Blue). Las componentes de cada color primario en el color resultante se expresan con números enteros entre 0 y 255, siendo 0 la intensidad mínima de ese color y 255 la máxima.

Hemos seleccionado 8 colores distintos para las líneas de autobuses y metro, en los casos en los que se introducen más de 8 líneas en el sistema se repiten los colores con los que se colorean las líneas.

Estos 8 colores que hemos seleccionado son: azul, rojo, amarillo, verde,

magenta, naranja, rosa y cian. Las líneas de autobuses y metro tienen asignados uno de estos colores según el orden de introducción.

Para diferenciar en una misma línea el tramo de ida del tramo de vuelta el método que hemos seguido es representar la línea de vuelta con un color algo más claro que el de la línea de ida. Para obtener estos colores más claros hemos usado una función que hemos realizado para mezclar colores, en este caso lo lográbamos mezclando el color en cuestión con un color amarillo bastante próximo a blanco.

La función que hemos realizado encargada de mezclar dos colores es la siguiente:

```
public static Color mezclaColores(Color c0, Color c1) {

	double totalAlpha = c0.getAlpha() + c1.getAlpha();
	double weight0 = c0.getAlpha() / totalAlpha;
	double weight1 = c1.getAlpha() / totalAlpha;

	double r = weight0 * c0.getRed() + weight1 * c1.getRed();
	double g = weight0 * c0.getGreen() + weight1 * c1.getGreen();
	double b = weight0 * c0.getBlue() + weight1 * c1.getBlue();
	double a = Math.max(c0.getAlpha(), c1.getAlpha());

	return new Color((int) r, (int) g, (int) b, (int) a);

}
```

Tabla 9. Código función mezclaColores

Otro uso importante que le hemos dado a la función que mezcla colores ha sido para representar tramos por las que pasan más de una línea. Como hemos dicho antes cada línea tiene asignado uno de los 8 colores disponibles en el sistema, color que se usa para colorear los tramos por los que pasa. Cuando por un tramo pasa más de una línea lo que hacemos es crear un nuevo color para el tramo en cuestión, este nuevo color es la mezcla de todas las líneas que pasan por dicho tramo.

Para colorear las paradas las hemos diferenciado en dos grupos, paradas que son parada inicial o final de alguna línea y paradas intermedias. Tanto un grupo como otro han sido dibujados siguiendo la siguiente estructura.

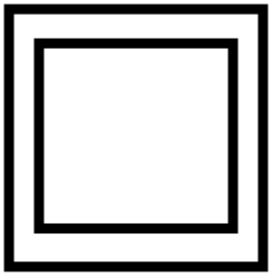


Imagen 6. Estructura parada

La diferencia que han presentado estos dos grupos ha sido a la hora de colorearse las paradas. En ambos grupos el cuadrado interior ha sido coloreado del color asignado a línea que para en esa parada, en aquellos casos en los que más de una línea para en la parada en cuestión el cuadrado ha sido coloreado con el color resultante de mezclar los colores asignados a dichas líneas. Lo que diferencia a los dos grupos de paradas es el color del borde exterior, en las paradas iniciales o finales de alguna línea se colorea de color negro mientras que en las paradas intermedias se colorea de color beis.

Para la representación de textos en la aplicación hemos usado el método *drawstring* al que se le pasa una cadena de texto y las coordenadas donde empezar a escribirlo. En los autobuses y vagones de metro se ha indicado el número de personas que van subidas en él. Cada vez que un autobús o vagón de metro sale de una parada se indica mediante el método recién indicado el número de personas que se han subido y bajado del autobús o metro en esa parada.

Para la realización de varios de los iconos que se han usado en esta aplicación se ha usado la aplicación Paint.net.

Imagen 7. Logo Paint.net

Lo mejor que presenta esta aplicación es que es bastante más potente que el famoso Paint de Windows pero no tan complejo como el también conocido Photoshop.

7 Control de errores

En el lenguaje Java, una excepción es un cierto tipo de error o una condición anormal que se ha producido durante la ejecución de un programa, aunque se puede dar una definición algo más sencilla, "*una excepción es un problema*"*.* En la programación siempre se pueden cometer errores que hay que gestionar de alguna forma e intentar solventarlos de la mejor manera posible, o dicho de otra forma que al usuario le incomode lo menos posible.

Para ello, Java cuenta con un mecanismo que consiste en utilizar una serie de bloques *try/catch/finally*. Para controlar las excepciones, lo que se hace es introducir el código que puede generar algún tipo de problema, como por ejemplo un error de entrada/salida o un error de índice fuera de rango, dentro de un bloque *try*, el cual se asocia a bloques del tipo *catch*. Precisamente dentro de estos bloques es donde se tratan las excepciones una vez que se producen.

Este tipo de manejo de excepciones es bastante eficaz para poder controlar los errores no deseados. Por otro lado, a medida que íbamos elaborando la aplicación decidimos definir secuencias de control para poder vigilar y controlar el comportamiento de nuestra aplicación ante las excepciones que fueran surgiendo, ya que estábamos interesados en que la aplicación respondiera de una manera entendible para el usuario cuando apareciera cualquier problema.

En nuestra aplicación debido a que es el usuario quien introduce todos los datos del sistema sobre el que se realizan los cálculos, hemos tenido que usar en numerosas ocasiones técnicas de control de errores, pues algo tan simple como introducir algo distinto a un número en algún campo reservado para ello puede hacer fallar a la aplicación completa.

A continuación vamos a ver algunos ejemplos de los errores controlados dentro de nuestro sistema.

Al iniciar la aplicación si seleccionamos introducir los datos del sistema manualmente lo primero que se nos pregunta es el número de líneas que van a existir en el sistema, si en vez de introducir un número introducimos algún carácter distinto o algún número menor o igual que 0, esto haría que se produjese un error debido que al intentar crearse una matriz del tamaño indicado no sería posible. En el siguiente fragmento de código se puede ver como mediante el bloque *try/catch* se controla que lo que se introduce es un número entero y mediante un *if* controlamos que sea mayor que 0. Gracias al bucle *while* hacemos que mientras no dejen de producirse excepciones se continúen pidiendo el número de líneas en el sistema.

```
jButton1.setText("Introducir datos");
       jButton1.addActionListener(new java.awt.event.ActionListener() {
       public void actionPerformed(java.awt.event.ActionEvent evt) {

              jButton1ActionPerformed(evt);
              setVisible(false);
              boolean ok = false;

            while (ok == false) {

                     try {
       Data.numLineas = Integer.parseInt(JOptionPane.showInputDialog(null,
"Número de líneas","Simulador red autobuses",JOptionPane.QUESTION_MESSAGE))
* 2;

       if (Data.numLineas > 0) {
                                  ok = true;
       InterfazIntroducirDatos introDatos = new InterfazIntroducirDatos();
                           introDatos.setVisible(true);
       }

                      } catch (Exception e) {
                                         ok = false;
                          }
        }
                     }
              });
```

Tabla 10 .Código Java control errores

Otro caso en el que es necesario el control de errores es cuando cargamos los datos del sistema desde un fichero .CSV. Algunos de los errores que se pueden producir son, por ejemplo, que seleccionemos para cargar un fichero de un tipo distinto a .CSV o que los datos introducidos no respeten el formato establecido. Cuando algo de esto se produzca en el bloque *catch* se lanzara un mensaje de error como el de la imagen siguiente.

Imagen 8. Interfaz mensaje error datos

8 Manual de usuario

8.1 Instalación

La instalación de la aplicación requiere de la copia los archivos que la componen en el equipo en el cual se pretenda instalar. Estos archivos se encuentran en la carpeta llamada *SimulacionAutobusesMetro* que se encuentra en el CD. Además hay que seguir los siguiente pasos:

- Instalar la máquina virtual de Java que se puede obtener gratuitamente en la página oficial de Java [7] y además está incluida en nuestro CD.
- Instalación del programa Maxima que se encuentra en el CD (Maxima 5.31.2 posterior) ya que será el encargado de realizar los cálculos.
- La edición del archivo "jacomax.properties" que se encuentra en la carpeta *SimulacionAutobusesMetro* donde indicaremos la ruta donde se encuentra el archivo *maxima.bat* que estará dentro de la carpeta *bin* que a su vez estará donde se encuentre Maxima instalado en el equipo.
- Colocar la carpeta llamada *paquetesMaxima* que se encuentra en el CD en la raíz del disco duro C.

8.2 Entrada de datos

La primera interfaz que muestra el simulador en cuanto lo abrimos es una que nos permite seleccionar de qué forma queremos introducir los datos.

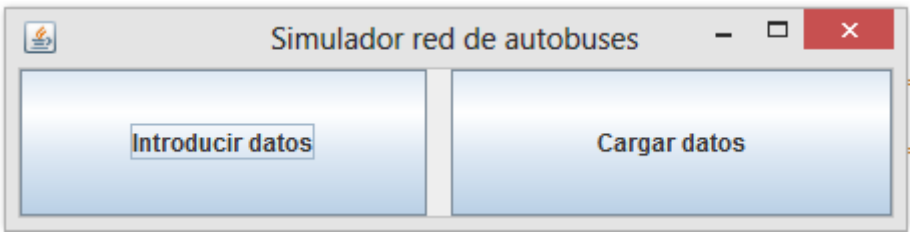


Imagen 9. Interfaz inicial autobuses

Como podemos ver existen dos formas introducir los datos que configurarán el sistema sobre el que se realizarán los cálculos, introduciéndolos de forma manual o cargándolos mediante un fichero de extensión .CSV.

8.3 Introducción de datos manualmente

Al pulsar el botón Introducir datos se habilita un pequeño panel como el de la siguiente imagen en el cual tenemos que introducir el número de líneas que deseamos que tenga nuestro sistema.

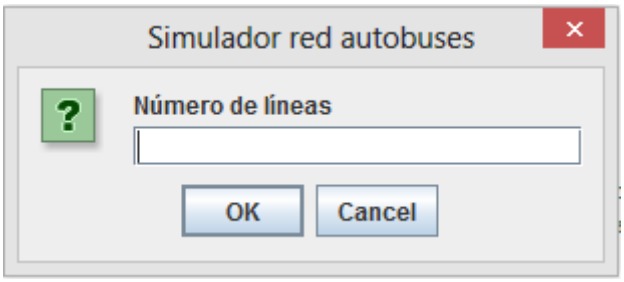


Imagen 10. Número líneas sistema

Una vez que se introduce el número deseado de líneas se genera una interfaz, como la de la siguiente imagen, en la cual tenemos que introducir el número de paradas que va a tener cada línea, diferenciando el sentido de ida del de vuelta.

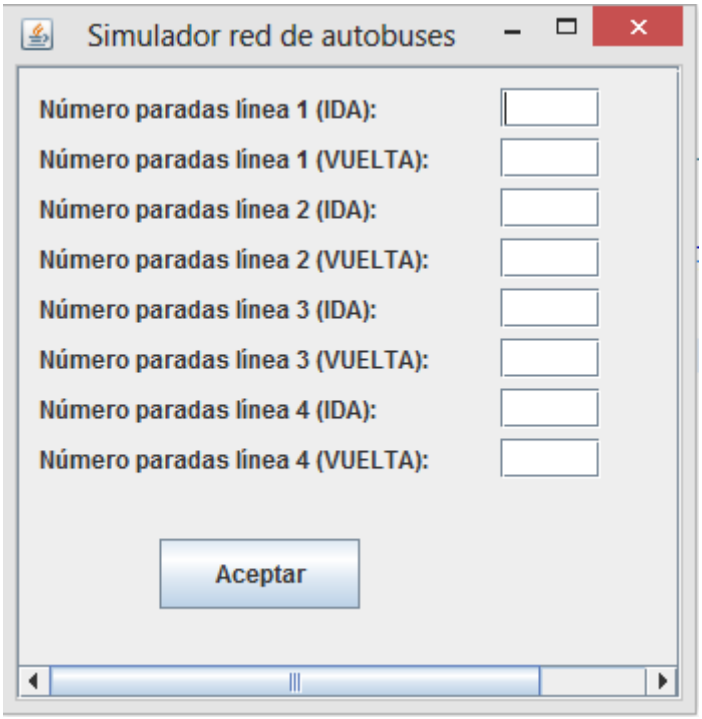


Imagen 11. Número paradas

Al pulsar el botón Aceptar se habilita una nueva interfaz en la cual podemos introducir todos los datos necesarios para crear el sistema deseado. Para un supuesto caso con 4 líneas la interfaz creada sería la siguiente:

Imagen 12. Introducción datos sistema

Este es otro caso en el que podemos ver que la aplicación se adapta a las necesidades del usuario, pues la interfaz se crea según los datos introducidos por él.

8.3.1 Lectura fichero CSV

Cuando optamos por la opción de cargar los datos del sistema desde un fichero con extensión .CSV se habilita un explorador mediante el cual podemos seleccionar el fichero deseado.

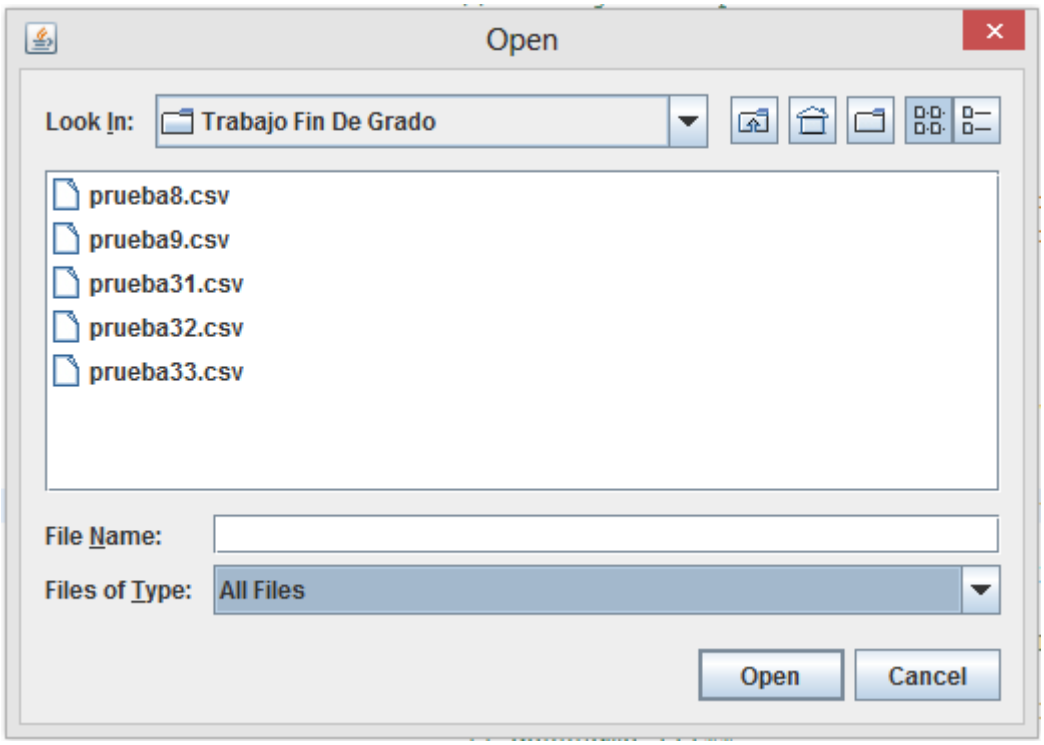


Imagen 13. Interfaz selección fichero CSV

El formato de entrada de datos en el archivo .CSV es el siguiente:

- En la primera línea, primera columna se indica el número de líneas del sistema.
- ******* ← *Indicador de cambio de tipo de dato a leer.*
- En las siguientes filas, tantas como líneas de autobuses o metro, se indican el número de paradas que tiene la línea en cuestión, en la primera columna en el sentido de ida y en la segunda columna en el sentido de vuelta.
- *** ← *Indicador de cambio de tipo de dato a leer.*
- A continuación vienen tantos conjuntos de dos filas como líneas de autobuses o metro introducidas. Estos conjuntos de dos líneas hay que introducirlos en el mismo orden que se introdujeron los números de paradas, es decir, en primer lugar se introduce el conjunto de la línea cuyo número de paradas se indicó en primer lugar, y así sucesivamente. Cada conjunto de dos filas, es decir cada conjunto de información de cada línea irá separado del siguiente por el símbolo *. En estos conjuntos de dos filas se indicará la siguiente información:

  En la primera fila, primera columna se indicará la frecuencia con la que pasan los autobuses por la línea en cuestión. En la segunda columna la hora de salida del primer viaje desde la primera parada de la línea de ida. El formato con el cual se introduce la hora es HHMM, es decir, siempre 4 números entre 0000 y 2359.
  A partir de esta columna se introducirán en orden las coordenadas de cada parada y separación en minutos con la parada siguiente, así hasta llegar a la última parada de la línea en el sentido de ida. Se introducirá primero la coordenada X y después la coordenada Y.
  En la segunda fila del conjunto, la primera columna se dejará vacía y

a partir de esa columna se introducirá la misma información que en la fila anterior, pero esta vez referente al sentido de vuelta.

- Una vez que hemos introducido todos los conjuntos con información de todas las líneas del sistema, indicamos el final del documento con:
  *****
  ****

Veamos un ejemplo para dejar más claro la forma de introducir datos a través de un fichero CSV. Supongamos que queremos introducir el siguiente mapa:

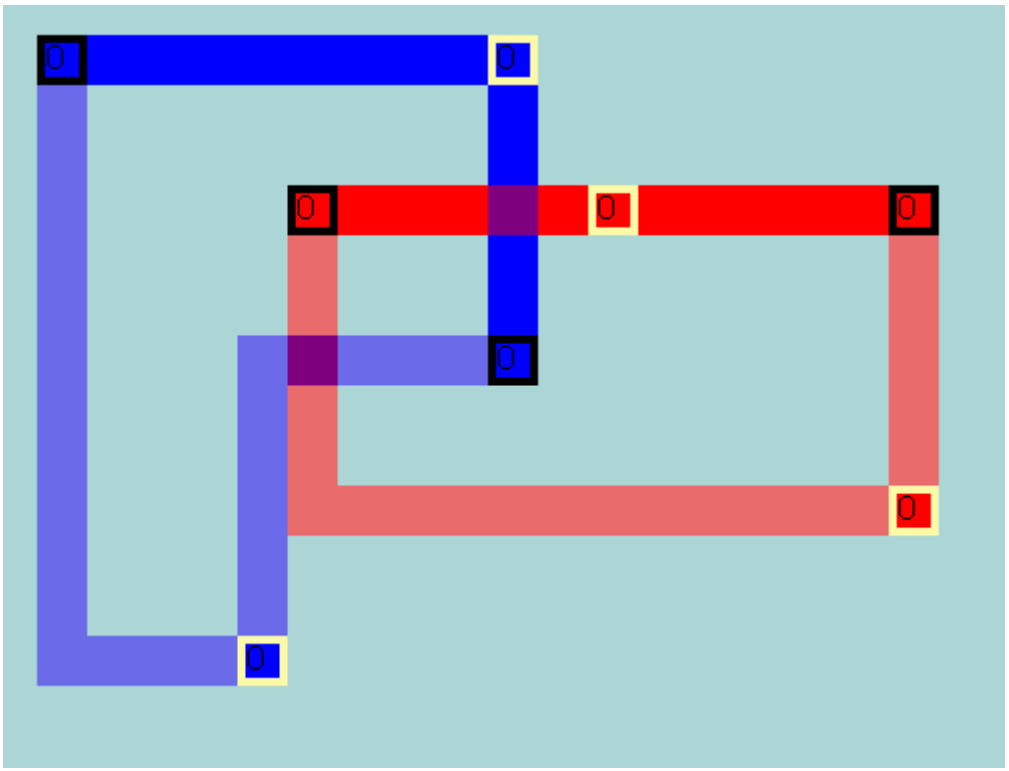
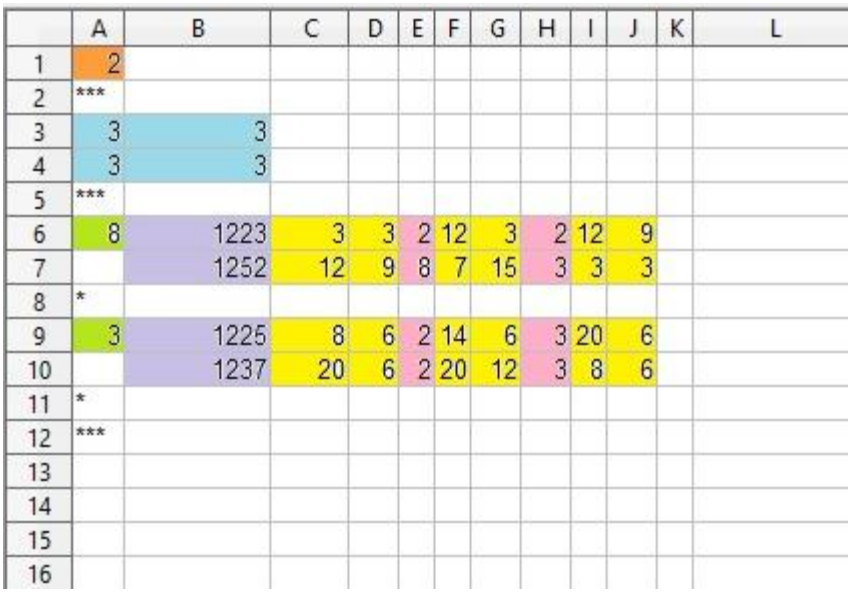


Imagen 14. Mapa ejemplo

Para ello necesitaremos un fichero CSV como el de la siguiente imagen, para hacerlo más intuitivo le hemos añadido colores a los distintos tipos de datos:



	A	B	C	D	E	F	G	H	I	J	K	L
1	2											
2	***											
3	3	3										
4	3	3										
5	***											
6	8	1223	3	3	2	12	3	2	12	9		
7		1252	12	9	8	7	15	3	3	3		
8	*											
9	3	1225	8	6	2	14	6	3	20	6		
10		1237	20	6	2	20	12	3	8	6		
11	*											
12	***											
13												
14												
15												
16												

Imagen 15. Ejemplo datos CSV

Se puede ver en la primera celda en naranja el número de líneas, en este caso 2. A continuación en azul aparecen indicadas el número de paradas de cada línea, primero en el sentido de ida y después en el sentido de vuelta, en este caso 3 en cada sentido de cada línea.

Después vienen los datos que caracterizan a cada línea, en grupos de 2 filas por cada línea, como en este caso tenemos 2 líneas entonces tenemos 2 grupos de 2 filas. En la primera celda en verde viene la frecuencia con la que pasan los autobuses o el metro por la línea en cuestión. Ahora cada fila muestra respectivamente para ida y vuelta, en primer lugar en morado la hora de la primera salida que se realiza en ese sentido. En amarillo podemos ver las coordenadas de las paradas y en rosa entre cada par de paradas los minutos que las separan.

8.4 Selección de parámetros

Una vez que han sido introducidos todos los datos que configuran el sistema sobre el que vamos a trabajar, ya se hayan introducidos manualmente o mediante un fichero CSV, se activará una interfaz como la de la imagen siguiente que nos permite determinar los valores de 4 parámetros claves para el funcionamiento del sistema.

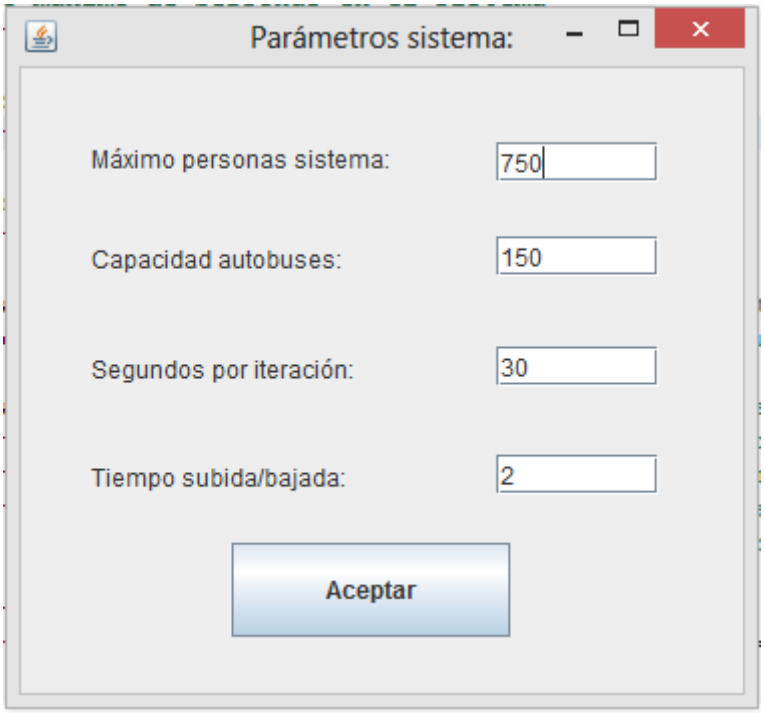


Imagen 16. Interfaz parámetros sistema

El primero de estos 4 parámetros como su propio nombre indica es el máximo de personas que puede haber en el sistema. No quiere decir que por el sistema no vayan a pasar más personas que el número indicado, se refiere al máximo número de personas que podrá haber en el sistema al mismo tiempo.

El segundo parámetro que podemos modificar es la capacidad de los autobuses, lógicamente en un autobús nunca subirán más personas cuando se sobrepase su máximo establecido.

El siguiente parámetro indica los segundos que va a simular Maxima en cada iteración. Por ejemplo, para el caso indicado en la imagen anterior el reloj global del sistema aumentará 30 segundos en cada iteración. El número máximo permitido es 59.

El último parámetro que podemos modificar en esta interfaz es el tiempo que van a tardar las personas en subirse o bajarse de un autobús o metro, este tiempo son segundos. Cada vez que se suban o bajen personas de un autobús o metro se sumara el tiempo correspondiente al tiempo que va a tardar el autobús en realizar su siguiente tramo, con lo que podremos ver que en situaciones en las que existan grandes cantidades de personas en las paradas las posibilidades de que se produzcan retrasos aumenta.

Cuando pulsemos el botón aceptar la siguiente interfaz que vamos a ver va a ser la que nos indicará el número de autobuses o vagones de metro con los que se ha inicializado el sistema. En caso de desear un número distinto de autobuses pulsaremos el botón Modificar.

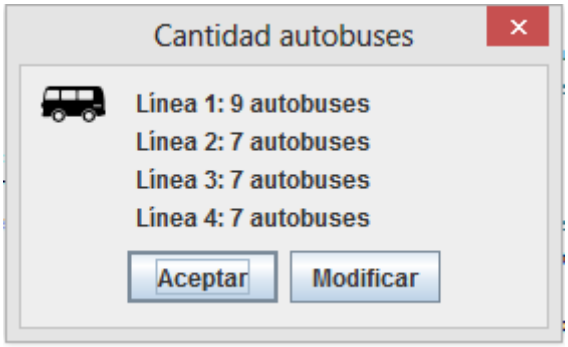


Imagen 17. Interfaz autobuses predefinidos

Al pulsar el botón Modificar una interfaz como la de la siguiente imagen nos permite determinar el número deseado de autobuses para cada línea.

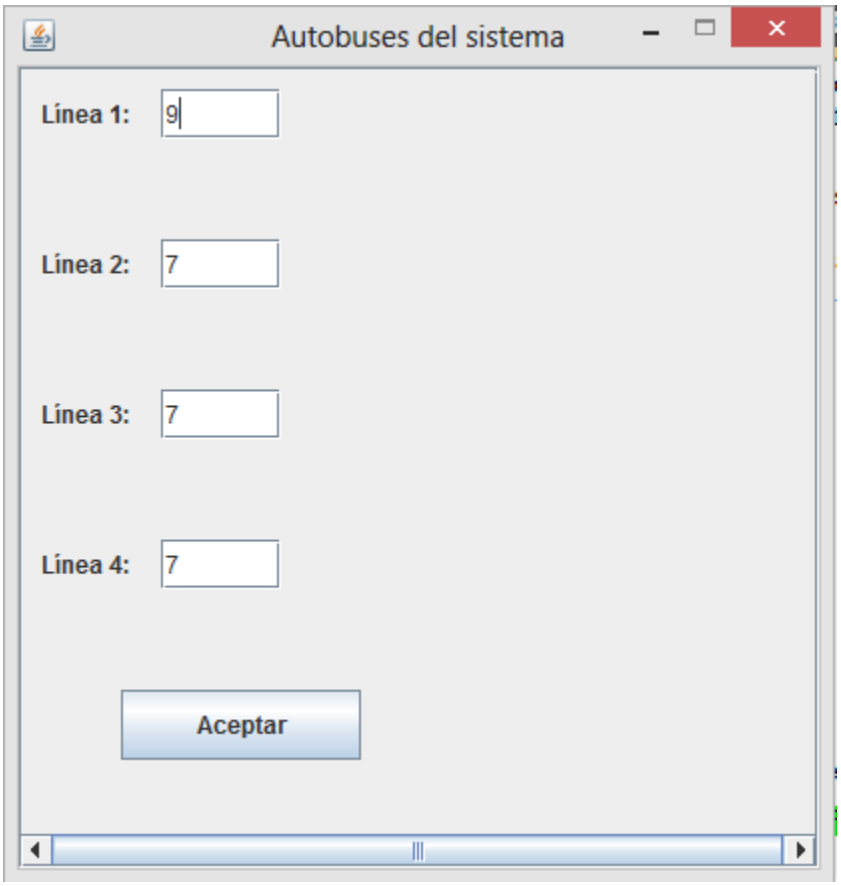


Imagen 18. Modificar autobuses

Una vez que aceptamos el número de autobuses y vagones de metro, ya sea el ofrecido por el sistema o el indicado por el usuario, se inicia la pantalla principal de nuestra aplicación.

8.5 Pantalla principal

En la siguiente imagen podemos ver un ejemplo del aspecto de la aplicación para un conjunto de líneas determinado. El fichero .CSV a partir del cual se ha generado el siguiente sistema se muestra en el anexo.

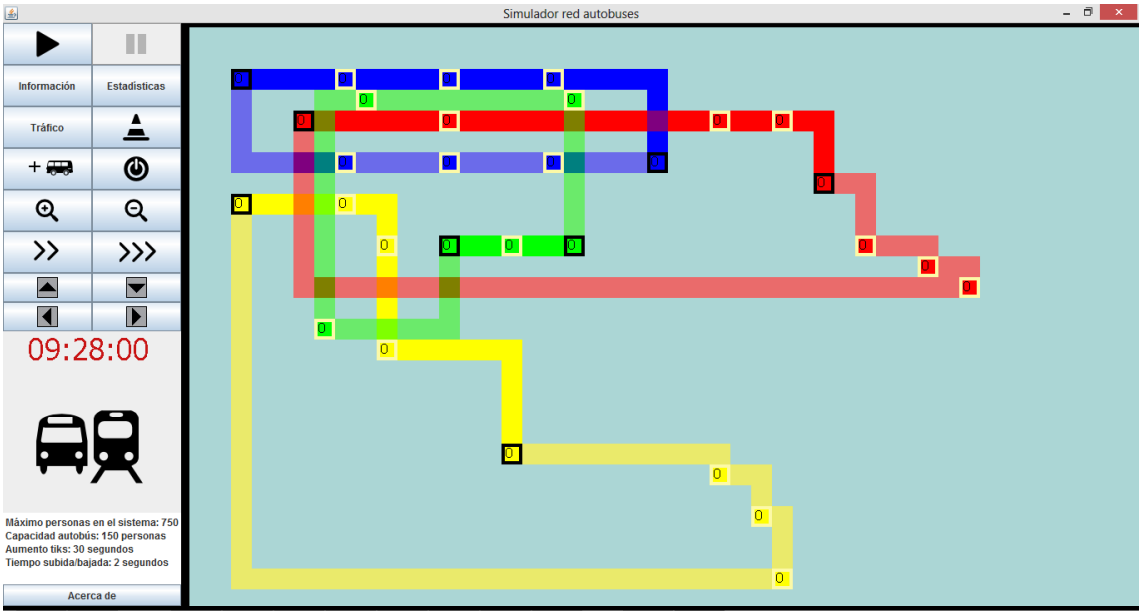


Imagen 19. Menú principal

8.5.1 Menú lateral izquierda

: Representa la hora en la que se encuentra el sistema.

A continuación se hace una breve descripción de la función de cada uno de los botones que presenta la aplicación:

-  : Inicia la simulación.
-  : Pausa la simulación.
- 

: Muestra en una interfaz como la de la siguiente imagen información sobre los parámetros del sistema que está cargado.

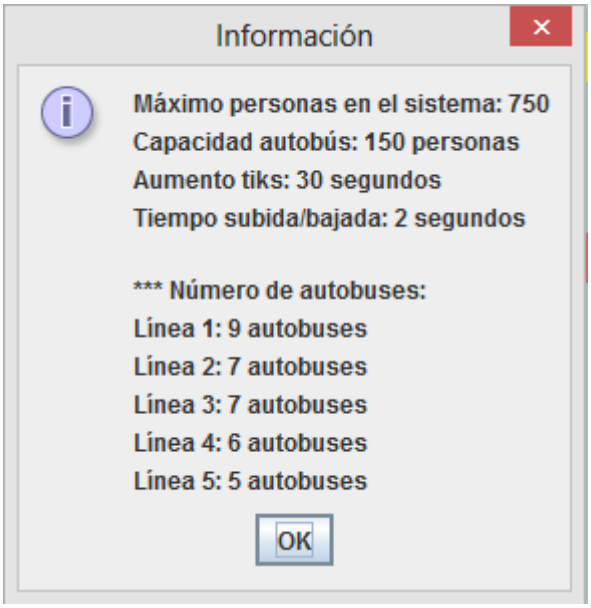


Imagen 20. Información sistema

- 

: Muestra las estadísticas del sistema en una interfaz como la de la siguiente imagen y genera un archivo del tipo .TXT y otros dos de tipo .CSV con las estadísticas registradas.

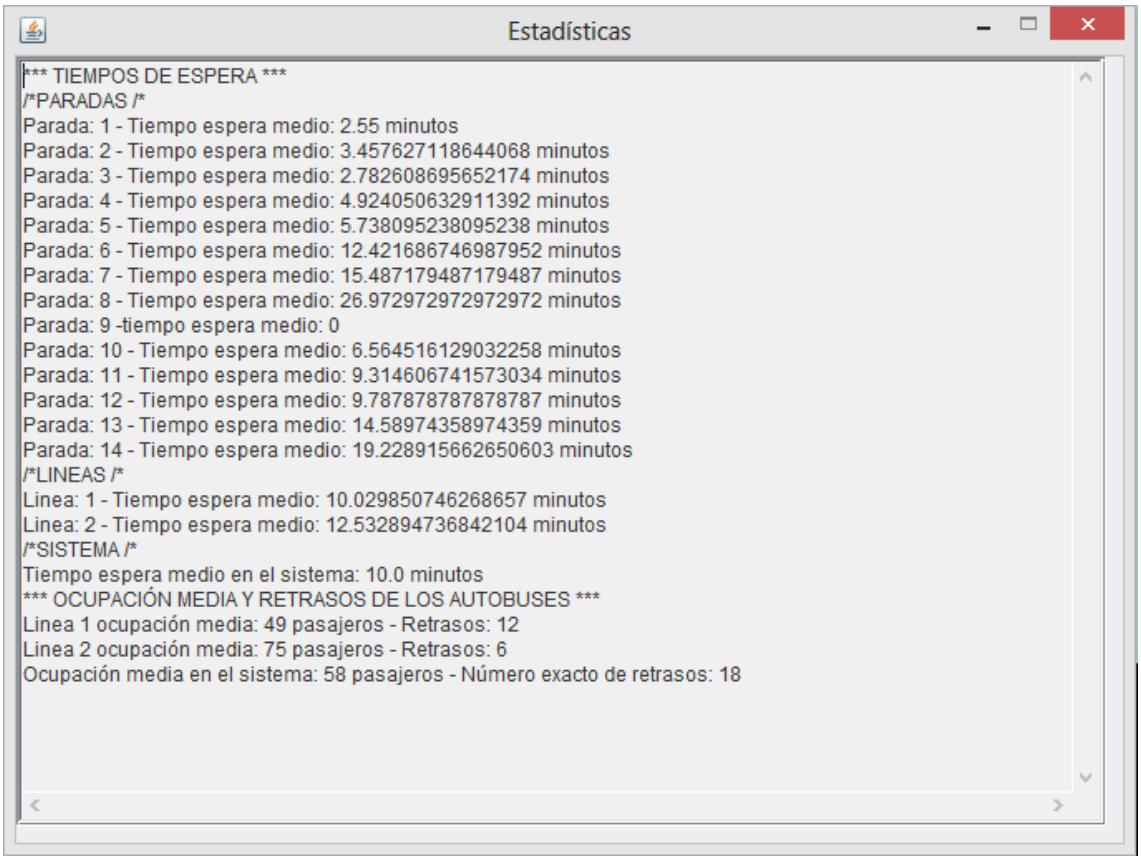


Imagen 21. Estadísticas

-  : Inicia la siguiente interfaz que permite editar los parámetros por los que se rigen las averías que se pueden producir en los autobuses o en los vagones de metro del sistema:

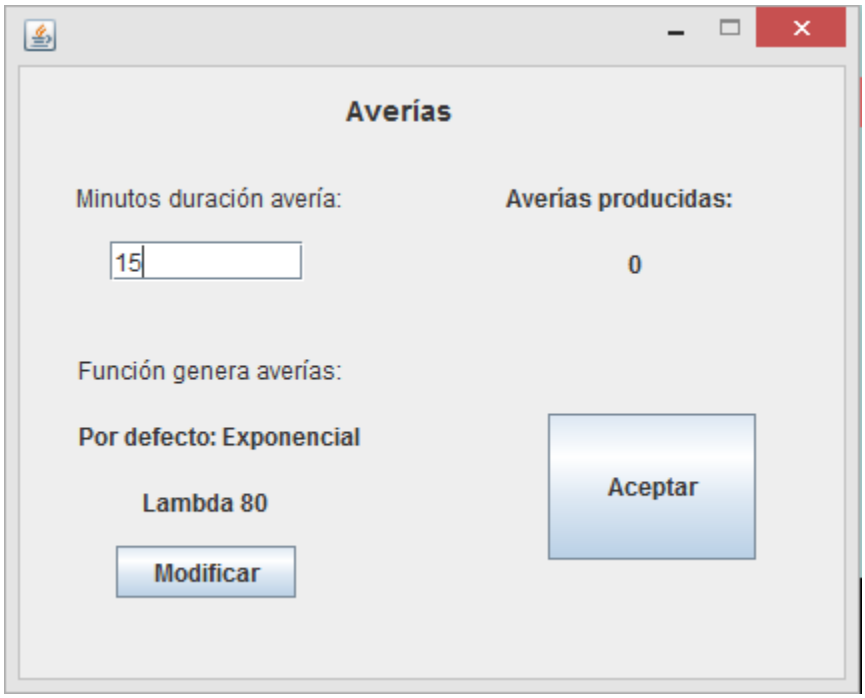


Imagen 22. Parámetros averías

En esta nueva interfaz si pulsamos alguno de los botones Modificar se inicia [8].

-  : Inicia un panel como el de la siguiente imagen que permite seleccionar a que línea queremos añadir un autobús o metro.

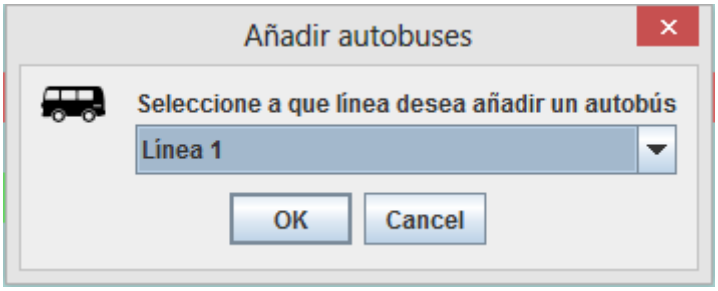


Imagen 23. Añadir autobuses

-  : Abre la siguiente interfaz:

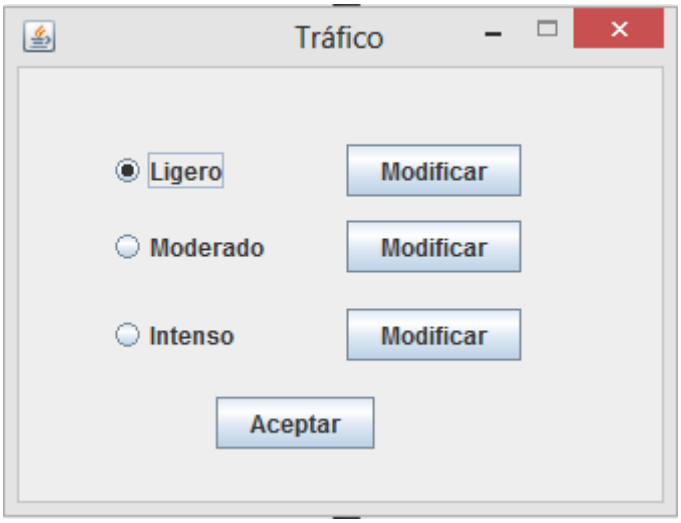


Imagen 24. Selección tráfico

En esta interfaz aparece seleccionado el tipo de tráfico que se encuentre establecido en el sistema en ese momento. Al pulsar en cualquiera de los botones Modificar a la derecha de los tipos de tráfico se abre una nueva interfaz como la de la siguiente imagen que permite editar los parámetros de las funciones que caracterizan al tipo de tráfico en cuestión.

Imagen 25. Editar tráfico

En esta nueva interfaz si pulsamos alguno de los botones Modificar se inicia [8].

-  : Aumenta el tamaño del mapa.

-  : Disminuye el tamaño del mapa.
-  : Sale del sistema registrando las estadísticas.
-  : Aumenta x2 la velocidad inicial.
-  : Aumenta x3 la velocidad inicial.
-  : Desplaza el mapa hacia arriba.
-  : Desplaza el mapa hacia abajo.
-  : Desplaza el mapa hacia la derecha.
-  : Desplaza el mapa hacia la izquierda.
- 

: Muestra la siguiente interfaz:

Imagen 26. Interfaz Acerca de

9 Conclusiones

Tras el trabajo expuesto en los apartados anteriores, podemos afirmar que se ha cumplido el objetivo inicial tener una aplicación que nos permite simular el tráfico de una red metropolitana de autobuses y de metro en circunstancias muy variables. Cabe destacar en este punto la gran ventaja de las simulaciones en tiempo acelerado (en precio, recursos, tiempo empleado ...) frente a las simulaciones físicas, en las que precisamente en nuestro caso el gasto en recursos se dispararía y en muchas ocasiones sería directamente imposible.

Estas simulaciones pueden ser usadas para dos propósitos distintos: como ayuda durante el establecimiento de nuevas líneas de autobuses o metro y para analizar el impacto de un posible cambio en unas líneas ya establecidas.

También hay que destacar la importancia de contar con una interfaz gráfica que produzca la información visual necesaria para saber qué ocurre durante la simulación, cosa que resulta bastante útil a la hora de visualizar los efectos de los cambios que se hagan en los parámetros de entrada.

Uno de los principales problemas que hemos encontrado en el desarrollo de esta aplicación ha sido la depuración de programas en Maxima, ya que Maxima carece de herramientas que nos ayuden en dicha tarea, como mucho, lo único que hace cuando se produce un error es dar una mensaje de error indicando de que tipo se trata, pero por ejemplo, cuando el error que se producía era el acceso a una posición errónea de una lista entonces era bastante tedioso encontrar la zona donde se producía debido a que en la mayor parte del código se realizan accesos a listas.

En vista al futuro sería interesante modificar la aplicación [2] usando el algoritmo descrito en 2.4 pues con eso se lograría que la aplicación tuviese un mayor campo de posibilidades al poder realizar pruebas con distintos mapas como sucede con esta aplicación. Otra posibilidad algo más ambiciosa sería hacer una nueva aplicación que combinase esta que hemos realizado con [2], es decir, que permitiese simular absolutamente todo el tráfico de una ciudad, tanto privado como público.

Referencias bibliográficas

[1] ALMIRÓN GARCÍA, ADRIÁN. (2012). Simulación del recorrido del equipaje en un aeropuerto con MAXIMA. Proyecto Fin de Carrera dirigido por Gabriel Aguilera Venegas y José Luís Galán García.

[2] CAMPOS DÍAZ JOSÉ CARLOS. (2014). Simulación de tráfico con semáforos y señales inteligentes usando un CAS. Proyecto Fin de Carrera dirigido por Gabriel Aguilera Venegas y José Luís Galán García.

[3] MCKAIN, D., (2012). Jacomax (Java Connector for Maxima) [online] Disponible en: https://www.wiki.ed.ac.uk/display/Physics/jacomax

[4] JULIA GARCÍA GALISTEO, MARÍA DEL CARMEN MORCILLO AIXELÁ, MANUEL RUÍZ CAMACHO. Curso de probabilidad y estadística.

[5] Documentación de Java http://docs.oracle.com/javase/7/docs/api/

[6] Manual Maxima : http://maxima.sourceforge.net/docs/manual/es/maxima.html

[7] Máquina virtual de Java http://www.oracle.com/technetwork/java/javase/downloads/jdk8-downloads-2133151.html?ssSourceSiteId=otnes

[8] RAMÍREZ LÓPEZ, MANUEL (2014). Mejoras en los proyectos de simulación de flujo de tráfico en tiempo acelerado. Trabajo fin de grado dirigido por Gabriel Aguilera Venegas y José Luís Galán García.

Anexos

Fichero CSV que genera el mapa visto en la imagen 19.

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R
1	4																	
2	***																	
3	5	5																
4	5	5																
5	5	5																
6	3	5																
7	***																	
8	5	930	2	2	5	7	2	5	12	2	5	17	2	5	22	6		
9		1000	22	2	5	17	6	5	12	6	5	7	6	5	2	2		
10	*																	
11	5	931	5	4	2	12	4	5	25	4	6	28	4	3	30	7		
12		1001	30	7	3	32	10	6	35	11	3	37	12	3	5	4		
13	*																	
14	5	931	2	8	5	7	8	5	9	10	5	9	15	3	15	20		
15		1000	15	20	3	25	21	3	27	23	2	28	26	4	2	8		
16	*																	
17	5	935	12	10	5	15	10	5	18	10								
18		1002	18	10	5	18	3	5	8	3	5	6	14	5	12	10		
19	***																	
20																		